Energy-Efficient On-Chip Cache Architectures and Deep Neural Network Accelerators Considering the Cost of Data Movement

Hongjie Xu

March 3, 2021

Acknowledgments

I would first like to thank my thesis advisor Prof. Hidetoshi Onodera. The door to Prof. Onodera office was always open whenever I ran into a trouble spot or had a question about my research or writing. Prof. Onodera consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed it.

I would like to thank Profs. Takashi Sato, Eiji Oki, and Sadao Kurohashi for valuable comments that greatly improved the manuscript.

I wish to express my sincere appreciation to my supervisor, Prof. Tohru Ishihara, who convincingly guided and encouraged me to be professional and do the right thing even when the road got tough during my master course. Without his persistent help, the goal of this PhD thesis would not have been realized.

I would also like to thank Shiomi sensei who was involved in the validation survey for this research project for introducing me to the topic as well for the support on the way. Without their passionate participation and input, the validation survey could not have been successfully conducted.

The physical and technical contribution of “WISE Program, Innovation of Advanced Photonic and Electronic Devices” is truly appreciated. Through the program, I have learned a lot about research methodology and various interdisciplinary knowledge. Without their financial support and funding, I would not have been able to graduate in time.

Finally, I must express my very profound gratitude to my parents and to my labmates for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Hongjie Xu
in Kyoto
March 3, 2021

Abstract

Energy-Efficient Cache Architectures and Deep Neural Network Accelerators Considering the Cost of Data Movement

by

Hongjie Xu

Doctor of Philosophy in Informatics

Kyoto University

Professor Hidetoshi Onodera, Chair

In data processing, the cost of data movement often dominates the hardware performance. Based on factors that affect the prediction capability of the cost of data movement, this thesis investigates representative types of processing tasks as general-purpose processing and special-purpose processing such as Deep Neural Network. This thesis discusses data properties that influence data movement in each processing task. Taking advantage of the proposed properties, this thesis designs four hardware designs to reduce the cost of data movement for various processing tasks.

For general-purpose processing which has a completely unpredictable data access pattern, this thesis proposes an energy-efficient hybrid cache memory system to exploit the non-linear relationship between the cache miss rate and the cache capacity. As the cache capacity increases from 0, the cache miss rate decreases rapidly as the capacity rises because there is no enough space to store all the data that is expected to be accessed in the near future by CPU. When the cache capacity exceeds a certain capacity, the cache miss rate decreases at a slower rate. According to the non-linear relationship, a hybrid 2-level on-chip cache architecture is first introduced as a replacement of a normal SRAM cache architecture to save the energy consumption. This thesis then discusses a method for finding the best mix of Standard-Cell Memory and SRAM, which minimizes the energy consumption of the hybrid cache under a cache area constraint. The simulation result shows a hybrid 2-level cache architecture by our method reduces the energy consumption by 68% in our case study of an instruction memory subsystem without increasing the die area.

In Deep Neural Network, which has a large data scale but a regular access pattern, this thesis introduces hardware properties to describe the precise cost of data movement in each memory hierarchy. Based on hardware properties, this thesis proposes a DNN accelerator for large-scale processing with a regular access pattern in DNN and a set of evaluation metrics that are able to evaluate the number of memory accesses and the required memory capacity according to a specialized processing dataflow. Proposed metrics are able to analytically predict the energy, the throughput, and the area of a hardware design without a detailed implementation. Once the processing dataflow and constraints of hardware resources are determined, the proposed evaluation metrics quickly quantify expected hardware benefits, thereby reducing design time.

In a sparse Deep Neural Network, an irregular data access pattern caused by sparsity brings great challenges to efficient processing accelerators. Focusing on the index-matching property in Deep Neural Network, this thesis aims to decompose sparse Deep Neural Network into easy-to-handle processing tasks to maintain the utilization of processing elements. According to the proposed sparse processing dataflow, this thesis proposes an efficient hardware accelerator called MOSDA to deal with the irregular data access pattern in sparse DNN. MOSDA can be effectively applied for operations of convolutional layers, fully-connected layers, and matrix multiplications. Compared to state-of-the-art neural network accelerators, MOSDA achieves 1.1× better throughput and 2.1× better energy efficiency than Eyeriss v2 in sparse Alexnet in our case study.

Focusing on the relationship between the memory capacity and the number of required memory accesses in Binary Neural Network which has a small data scale and a regular access pattern, this thesis utilizes a hybrid memory system to reduce energy consumption. According to the proposed processing dataflow, this thesis proposes an efficient hardware accelerator for small-scale processing with a regular access pattern in BNN processing. The proposed BNN accelerator can be effectively applied for operations of convolutional layers, and fully-connected layers. Compared to state-of-the-art binary convolutional accelerators, the proposed architecture achieves 1.4× better energy efficiency than ChewBaccaNN in Alexnet in our case study.

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Energy-efficient Hardware Design

1.1.2 The Cost of Data Movement

In Big Data Era, with ever-increasing demands for Internet-of-Things (IoTs) endpoint devices, energy-efficient System-on-Chips (SoCs) are highly required [1]. Those endpoint devices, or sometimes called edge devices, are interconnected with each other, enabling autonomous processing and communication of gathered data from the target. Endpoint devices employ not only multiple sensors, but also a small processor, which is mainly used for processing near-sensor data gathered by sensors. Since most endpoint devices are battery-powered, edge devices with an energy-efficient processor are highly required.

Inside a processor, the energy of data movement between memories is expected to be two orders of magnitude higher than the energy of double-precision floating-point operations [2]. As a result, the cost of data movement often dominates the energy consumption of an entire digital hardware processing system [2, 3]. The cost of data movement describes the energy consumption of data transmission between memories. The cost of data movement in a hardware architecture is determined by two factors: the number of memory accesses and the access energy to memories. In memory designs, the access cost is directly related to the memory capacity. We can evaluate the cost of data movement required to execute a processing task, once the target processing task and the corresponding hardware structure are determined. Considering a specified processing task such as a Deep Neural Network (DNN) processing, Ref. [4] proposes a property called *data reuse* to expose the precise number of memory accesses, which is to describe the number of accesses to the same data during the processing. Based on data reuse, previous works [5–9] build evaluation models to estimate the cost of data movement. However, data reuse is difficult to describe another important factor for data movement, which is the required memory capacity. As a result, data reuse is difficult to help design accelerators to reduce the cost of data movement.

1.1.3 Memory Hierarchy Design

In order to estimate the cost of data movement, this thesis investigates the memory architecture at first. In a modern processing architecture, the memory hierarchy separates a memory system into a hierarchy. The memory hierarchy uses a memory storage layer based on different access speed to memory data [10]. For the memory hierarchy in high-performance computers, the highly requested data is stored in a high-speed memory, allowing a central processing unit (CPU) core for faster access. The traditional memory hierarchy design is to accelerate operations by exploiting the locality of reference [11] to reduce the cost of data movement. An on-chip memory hierarchy acts as the buffer between CPU and main memory, which is used to hold parts of data which are expected to be accessed by CPU in a near future. As a result, many researchers have designed a memory architecture for CPU acceleration based on the locality of reference to reduce the number of accesses to time-consuming memories.

The dynamic energy consumption of a memory is also directly proportional to the number of memory accesses. If we focus on improving the energy efficiency of a single access at a memory level with a high number of accesses, we can also expect to reduce overall energy consumption at a low cost.

The energy of data movement can be deduced from the memory capacity and the number of memory accesses in a memory hierarchy. However, the target processing task sometimes makes it difficult to accurately evaluate the number of memory accesses and the memory capacity in a memory hierarchy. First, an irregular data access pattern makes it difficult to make an accurate assessment of the required memory accesses. Second, the data scale of a processing target affects the difficulty of the reduction for data movement on-chip. If the data scale is small enough that all data can be stored on-chip, we can reduce the movement for all data during processing. If the data size is too large, there will be an inability for on-chip memories to keep all data. Therefore we may be forced to make trade-offs and can only partially reduce the cost of on-chip data movement. This leads to different hardware processing strategies for small-scale processing and large-scale processing.

1.1.4 Processing Task Classification for Data Movement Evaluation

If we know precisely the memory capacity and the required number of memory accesses at each memory level for the target processing, we can estimate the cost of data movement required accurately and design the energy-efficient hardware. However, the difficulty and feasibility of predicting the cost of data movement vary in different processing tasks.

The data access pattern determines the feasibility of the evaluation of the cost of data movement in a target processing task. If the data access pattern is regular and fixed, we accurately evaluate the cost of data movement. If the data access pattern within the task is unpredictable, we can hardly make an accurate evaluation for the required number of memory accesses.

A hardware architecture is considered to handle an unpredictable data access pattern of processing tasks if it is designed to be a general-purpose processing hardware architecture. In this case, we are able to obtain performance gains to exploit the locality of reference which is a statistical property to describe that the processing unit tends to access the same memory location repetitively in a short time.

If the hardware architecture is designed to execute a particular type of processing task such as DNN processing, energy-efficient hardware designs could be designed to exploit a processing property inherent to a regular data access pattern. This thesis proposes DNN accelerators to exploit data properties in DNN processing. Since the access pattern is fixed and regular, we can build an evaluation model to accurately calculate the cost of data movement under different hardware structures. However, due to the large data size of DNN processing, it is often difficult to design a hardware architecture that stores all data to reduce the cost of data movement on-chip. Therefore, we need to design an evaluation model to guide us to make the trade-off between the reduction of the on-chip memory capacity and the reduction of off-chip memory accesses.

Different from dense DNN, sparse DNN has an irregular data access pattern due to its data sparsity. Previous DNN acceleration structures often need to skip zero-related operations in order to efficiently exploit the data sparsity to save energy. The unpredictable zero position leads to an irregular data access pattern. Fortunately, the irregular access pattern of sparse DNN is not completely unpredictable as in the case of general-purpose processing. We can decompose a sparse processing task into small tasks with a regular access pattern by a well-designed processing dataflow. As a result, we can design an energy-efficient hardware accelerator based on its data movement evaluation model.

Binary Neural Network (BNN) is with a regular access pattern and also a small data scale. For BNN processing, it is possible to design an energy-efficient memory hierarchy to reduce the cost of data movement, since all data can be stored on-chip [12]. As a result, we have an opportunity to design an energy-efficient memory hierarchy design that matches processing properties.

1.2 Motivation

This section introduces motivation in this thesis. Section 1.2.1 introduces motivation in Chapter 2, which targets data movement reduction in general-purpose processing. The first part of section 1.2.2 introduces motivation in Chapter 3, which targets data movement reduction in accelerators in DNN processing. The second part of section 1.2.2 introduces motivation in Chapter 4, which targets data movement reduction in sparse DNN processing. The third part of Section 1.2.2 introduces the motivation of Chapter 5, which focuses on data movement reduction of accelerators in BNN processing.

1.2.1 Multi-Level Cache Architectures for General-Purpose Processing

In general-purpose processing, since there is no way to know what a task to be processed will be, the data access pattern for processing will be unpredictable. Considering an irregular data access pattern, employing caches in hardware designs is one of the most effective approaches to improve both energy efficiency and performance [13–15]. Therefore, a cache architecture is implemented into most commercial processors. The on-chip cache architecture is a major component in edge processors that dominates the entire energy consumption and performance. Ref. [3] shows that a memory hierarchy including caches consumes half of the total energy consumption of embedded processors. While these nominal on-chip caches significantly improve processing performance, they also occupy a large portion of chip area [16, 17]. Therefore, achieving cache architectures with high energy and area efficiency is a must.

A key insight is that the relationship between the cache miss rate and the capacity is non-linear, which means that when a cache capacity increases, the miss rate usually drops sharply first and then converges. The implication is that the energy consumption in an overall memory hierarchy can be reduced by replacing a high-density memory with a low-density yet more energy-efficient memory under a given area constraint. Replacing the entire memory with an energy-efficient low-density memory will result in a significant loss of the cache capacity, which is due to an unacceptable miss rate of the on-chip cache. Therefore, we consider inserting a small Level 0 (L0) cache between the CPU and the Level

1 (L1) cache. Since a small cache has a short critical path, related works utilize the L0 cache to achieve high-speed processing [18–21]. Different from previous works, this thesis attempts to reduce the overall energy consumption of memory access by implementing the energy-efficient memory in a small but frequently accessed L0 cache.

1.2.2 Hardware Designs for Special Purpose Processing

In general-purpose processing, it is often difficult to specify the access patterns for each data. This unpredictability makes it difficult to design an energy-efficient hardware which reduces the cost of data movement. The data access pattern in a special-purpose processing task is often explicit. As a representative example of special-purpose processing, the processing hardware of DNN typically uses massively parallel processing to increase the throughput of DNN accelerators. Massively parallel data movement poses two challenges for hardware designs. First, large-scale data movement in DNN processing would pose a significant energy challenge. In order to adapt to a battery-powered terminal processing environment, we should utilize data properties to minimize the cost of data movement. Second, large-scale parallel processing will also bring challenges to the scheduling of parallel data transmission. It poses a challenge for control logics of cache designs to decide where each data will be distributed in parallel at which time. As a result, many DNN accelerators abandon the use of cache designs in favor of scratchpad memories for data movement in DNN processing [22–35].

Considering the factors that affect the prediction capability for the data movement, DNN processing tasks can be further classified into several tasks. First, the data access pattern is considered to determine the feasibility of an evaluation of the cost of data movement in the target DNN. If the data access pattern is regular and fixed such as dense DNN processing, the cost of data movement can be accurately evaluated. If the data access pattern within a parallel processing task is irregular such as sparse DNN, it is hard to estimate the required number of memory accesses accurately. Second, the data scale of a processing task is considered to affect the evaluation of the number of memory accesses and the memory capacity. If the data scale of a target program such as BNN processing is small enough that all data can be stored on-chip, the memory structure required for data processing is usually simple and clear. If the data size of a target program is larger than an on-chip memory storage capability, the data movement is more complex and difficult to be precisely evaluated. As a result, this thesis divides DNN processing tasks into dense DNN, sparse DNN, and BNN for a more accurate estimation of data movement.

DNN Accelerator for Large-Scale Processing with Regular Access Pattern

A DNN hardware accelerator relies on parallel processing of multiplications and additions for throughput enhancement. In order to evaluate the precise data movement during DNN processing, Ref. [27] introduces a detailed concept of data reuse of input feature maps (ifmaps), weights, and output feature maps (ofmaps). The number of data reuse is defined as “the number of multiply-and-accumulations (MACs) that use the same piece of data”. More specifically, the data reuse of an ifmap pixel is defined by the number of occurrences of the same ifmap pixel in DNN processing. The data reuse of a weight is defined similarly. The number of data reuse of an ofmap pixel is defined as the number of product terms in one ofmap pixel in DNN processing [4]. However, since data reuse describes the number of memory accesses but not the access cost to each memory hierarchy, it is difficult for data reuse to directly compare the cost of data movement in different hardware architectures. This thesis proposes properties that can be used to describe the number of memory accesses and the memory capacity in DNN processing that can accurately describe the memory hierarchy.

Sparse DNN Accelerator for Irregular Data Access Pattern

The sparsity of weights and ifmaps can be utilized by gating or skipping operations, thereby improving energy efficiency. In order to further enhance throughput by sparsity, a complex control logic is usually inevitable according to the irregularity of weights and ifmaps in sparse DNN. For achieving high Processing Element (PE) utilization, we will propose a method that aims to directly infer the number of operations from the number of loaded non-zero ifmap pixels and the number of loaded non-zero weights so that we can easily arrange weights and ifmap pixels required for massively parallel processing. We should fetch just the right amount of input data, thereby maximizing PE utilization with a simple control logic. This thesis focuses on data dependency between ifmaps and weights to evaluate its impact on the number of required operations for sparse DNN.

BNN Accelerator for Small-Scale Processing with Regular Access Pattern

By reducing a 32-bit floating-point arithmetic to a binary arithmetic, BNNs achieve significant energy reduction and performance improvements over traditional Convolution Neural Networks (CNNs). Unlike general-purpose processing, data access patterns in BNNs are regular. The data scale in BNN is also usually small enough that all data can be stored on-chip. Therefore, this thesis proposes a hardware design of an energy-efficient on-chip memory system for data in BNNs.

This thesis adopts the Near-Threshold (NT) memory structure [18], which is more energy-efficient than SRAM, as one level of an on-chip memory hierarchy to reduce the overall energy consumption. Considering the area overhead of an NT memory, we integrate a larger SRAM-based buffer, which is more area-efficient, to store all data for

BNN processing.

1.3 Thesis Contribution

This thesis aims to propose energy-efficient cache architectures and DNN accelerators by reducing the cost of data movement in various processing tasks. This thesis selects two representative processing tasks, which consist of general-purpose processing and DNN processing. This thesis discusses the processing properties that may affect the cost of data movement in each situation. Considering the cost of the data movement, this thesis designs four energy-efficient hardware designs to explore the effectiveness of data processing properties to guide the direction of future hardware designs.

In general-purpose processing, an energy-efficient hybrid cache architecture design is proposed to exploit the nonlinear relationship between the cache miss rate and the capacity. The simulation result shows that the Standard-Cell Memory (SCM) based design has an energy advantage over an SRAM-based design. Using SCM as the L0 cache, a 2-stage hybrid cache architecture exhibits better energy consumption than a conventional SRAM-based cache architecture in simulations.

In DNN processing, the data movement between memory and processing elements consumes most of time, energy, and area. At the same time, the number of operations in DNN processing is usually much larger than the scale of hardware resources. The key idea is that a whole processing task can be divided into easy-to-handle small tasks to meet the hardware sizes. Next, we focus on the small tasks for a fine-grained evaluation of the cost of data movement. In this thesis, two hardware properties are investigated to describe the required number of memory accesses and the memory capacity in each memory hierarchy. Based on the two properties, this thesis proposes a DNN accelerator for large-scale processing with a regular access pattern in DNN and a new set of hardware metrics to analytically estimate energy, throughput, and area from processing dataflows. Based on the analytical evaluation of various processing dataflows and various processing environments, the proposed evaluation metrics can effectively evaluate processing dataflows without a hardware implementation. Designers are able to use the proposed evaluation metrics to significantly reduce hardware design effort.

This thesis also addresses a PE utilization problem in sparse DNN processing and sparse DNN properties. The key idea is to directly infer the number of operations from the number of loaded non-zero ifmap pixels and the number of loaded non-zero weights. This thesis proposes an efficient hardware accelerator called MOSDA for an irregular data access pattern in sparse DNN. MOSDA does not require complex control logic in sparse

DNN processing and speeds up the processing by skipping zero data. According to the case study, MOSDA outperforms state-of-the-art CNN accelerators in sparse Alexnet.

This thesis further proposes a hybrid memory system with an energy-efficient memory architecture for small-scale processing with a regular access pattern in BNN processing. Based on dedicated dataflows for matrix multiplication and convolution, the proposed hardware achieves energy-efficient processing in DNN processing without crossbar logic for accumulating partial sums. The proposed architecture outperforms state-of-the-art DNN accelerators in the case study of this thesis.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents an energy-efficient cache architecture design based on properties that affect the hardware performance for general-purpose processing. Chapter 3 discusses properties that determine the cost of data movement in DNN processing. Based on the proposed properties, this thesis presents a dense DNN accelerator and an analytical evaluation model. Chapter 4 discusses the property that helps enhance the utilization of processing elements in sparse DNN processing. This thesis further presents a sparse DNN accelerator handling irregularity. Chapter 5 discusses the memory hierarchy which is efficient for BNN processing. Based on the proposed memory hierarchy, this thesis presents an energy-efficient BNN accelerator. Chapter 6 concludes this thesis.

Chapter 2

Hybrid Cache Memory for General-Purpose Processing

2.1 Introduction

In general-purpose processing, the data access pattern in a processing task is random. However, there is still a statistical feature to describe the cost of data movement. The principle of locality [11] is a property to describe that a processing unit tends to access one same memory location repetitively in a short time. The cache design is one of the most successful techniques to exploit the principle of locality. The hardware cache is a small memory close to the processor that is used to store recently referenced data or instructions to reduce the number of accesses to the external memory. CPU accesses the cache directly. External memory is accessed only when the CPU accesses the cache and data or instructions do not exist in the cache. The failed attempt to find data in the cache is called a cache miss.

Fig. 2.1 introduces the relationship between the cache capacity and the miss rate. When the cache capacity is increased from 0, the cache miss rate decreases rapidly since there is still no enough space to store all data which is expected to be accessed in the near future. When the cache capacity comes above a certain capacity, then the cache miss rate decreases slower [13]. As a result, the relationship between the cache capacity and the miss rate is non-linear [36, 37].

In a cache design, Level-0 (L0) caches have been proposed to improve performance [18–21]. A tiny L0 cache architecture is placed between the Level-1 (L1) cache and the CPU core. Since a small cache have a better access energy and an operating speed than a large cache, the performance of the entire system can be improved. Since the CPU core firstly accesses the L0 cache, the number of accesses for the energy-consuming L1

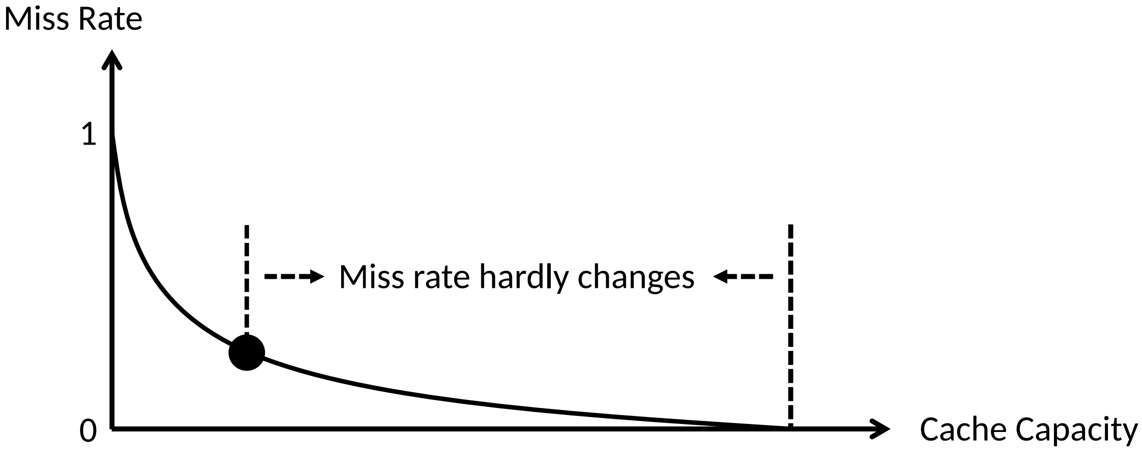


Figure 2.1: The relationship between the cache capacity and the miss rate.

cache can be effectively reduced with an acceptable area overhead [20].

The access cost of the L0 cache has a decisive impact on the performance of an entire memory system. Therefore, we should not only reduce the L0 cache capacity, but also reduce the access energy of the L0 cache by adopting more energy-efficient memory macros to replace conventional SRAM-based cache. Recently, the concept of Near-Threshold Computing (NTC) has been emerged [18]. It downscales the supply voltage from a nominal to a near-threshold voltage of MOSFETs. The dynamic energy consumption can be quadratically reduced by a scaling technique. A traditional 6 Transistor (6T) SRAM has a limitation to reduce energy since 6T-SRAM is hard to work at a near-threshold voltage. In order to maintain the robustness in the Near-Threshold Voltage (NTV) environment, memory macros usually use more complex peripheral circuits, or larger-sized transistors, or auxiliary power supplies for read and write operations [37–42]. Standard-Cell Memory (SCM) is a representative which has low processing speed and low area density, but inherent energy efficiency. Using SCM-based L0 cache can further reduce the overall power consumption of the cache architecture. The large area overhead is one major obstacle that prevents us from applying SCM to on-chip L0 memory. The result is that the SCM-based design has a lower cache capacity due to lower area density, and better energy performance with the same area constraint.

This chapter proposes a hybrid cache architecture which employs an energy-efficient SCM-L0 cache and an area-efficient SRAM-L1 cache. Based on the trade-off between cache capacity and energy efficiency, this chapter proposes a hybrid cache system with a filter cache architecture which employs an energy-efficient low-voltage memory and a conventional 6T-SRAM into the L0 cache and the L1 cache, respectively. Since the CPU core frequently accesses the energy-efficient L0 cache, the energy consumption can be effectively reduced. Since the L1 cache memory consists of 6T-SRAM with high area-density, it reduces the number of accesses for an energy-consuming off-chip memory. Compared with existing approaches on hybrid cache designs, the contribution of this chapter is summarized as follows:

- This chapter proposes a hybrid cache hierarchy where energy-efficient Near-Threshold memory and area-efficient 6T-SRAM are utilized as the L0 cache and the L1 cache, respectively.
- This chapter contributes a framework for selecting an appropriate cache architecture under given design constraints such as time and area constraints.
- For the first time, this chapter proposes an SCM-based L0 filter cache. This chapter shows that the SCM-based L0 cache achieves 68% less energy consumption compared with the SRAM-based L0 cache at near-threshold supply voltage in the simulation case of this chapter. Although NT-SCM has a better energy efficiency than NT-SRAM, NT-SCM is several times larger than NT-SRAM. However, due to the non-linear relationship between the miss rate and the cache capacity, the total energy consumption can be efficiently reduced by utilizing NT-SCM as the L0 cache. Applicable environments for the proposed hybrid cache are also analyzed.

The rest of this chapter is organized as follows. Section 2.2 presents a simple motivational example for the proposed hybrid memory structure. Section 2.3 presents a detailed architecture of the proposed hybrid cache hierarchy. Section 2.4 analyzes applicable environments for the proposed hybrid cache. Section 2.5 concludes this chapter.

2.2 On-chip Memory System for Ultra Low Energy Computing

2.2.1 Related Work

Several researchers focus on the low-power design of the L0 cache [18–21]. Ref. [21] uses a "victim" cache to reduce conflict misses and to further reduce the energy consumption. Its strategy is to insert a fully-associative cache between direct-mapped L1 cache and lower-level cache. In Ref. [20], a 128-byte cache is utilized as a filter cache in order to reduce the energy consumption at the cost of degrading the performance. Based on [21] and [20], Ref. [18] proposes a reconfigurable energy-efficient cache architecture for a near-threshold operation. The proposed architecture is a multi-way cache where an NT-SRAM and conventional 6T-SRAMs are utilized for one way and then the other ways, respectively. By swapping most frequently-accessed codes into a way composed of NT-SRAM, they obtain the energy reduction and maintain the performance at the same time.

At the same time, a number of researches on an NT-tolerant memory reduce energy by scaling a supply voltage down to a sub-threshold voltage [37, 38]. Some designs focus on improving SRAM read and write stability. A single-ended 6T-SRAM can operate stably below 200 mV at the cost of the area in comparison with the conventional 6T-SRAM [38]. A threshold-voltage (V_{th}) optimized SRAM which isolates a read path and a write path operate stably at a 300 mV supply voltage with an acceptable area overhead [39, 43]. In addition to enhancing the robustness of SRAM, many researchers also look for alternative memory structures to SRAM: Spin-Transfer Torque Magnetic Random-Access Memory (STT-RAM) demonstrates a possibility of replacing SRAM on Last Level Cache (LLC) in multi-core systems [40]. For convenience, this chapter calls 6T-SRAM and 10T-SRAM as 6T and 10T in the following, respectively.

2.2.2 Standard-Cell Memory

Nowadays, in order to guarantee a stable operation in the near-threshold region, SCM has been studied as an alternative to SRAM. SCM, which consists of standard cells only, operates even in sub-threshold regions [37, 41, 42]. The SCM has a fully digital structure utilizing latch cells as storage elements. Fully digital structures are implemented in read and write circuitry.

Inherently, SCM has a better dynamic energy efficiency than SRAM. A bit-line-based read operation with a strong sense amplifier in SRAM results in a large effective capacitance on bit-line in each readout/write operation. Since read operation is accompanied by charging and discharging the bitline in every cycle, SRAM energy consumption is larger than SCM without a bit-line structure. As a result, SCM exhibits better energy-efficiency even than 10T-SRAM. However, the SCM area is about 3 × larger than that of full-custom SRAM [37, 42].

Based on related researches above, this chapter uses a more energy-efficient but less area-efficient SCM structure than NT-SRAM as the L0 filter cache to reduce the energy consumption at a cost of slightly increasing a miss rate. This chapter focuses on the non-linear relationship between the capacity and the miss rate to further reduce the energy consumption of a total memory system.

2.2.3 Relationship between Miss Rate and Cache Capacity

An important point for this approach is that the relationship between the cache capacity and the miss rate is non-linear [36]. Therefore, the energy composition can be reduced by replacing SRAM with NT-SCM under a fixed area constraint even if the L0 capacity decreases to one third.

If we simply replace all on-chip memories with NT-SCM, designers suffer from a huge loss of the cache capacity. As a result, the processor has to access an energy-consuming off-chip memory more often. As a trade-off, this chapter places a small NT-SCM-based L0-filter cache between the L1 cache and the CPU. The purpose of the placement is to divide the on-chip cache architecture into two parts, so that we can optimize them separately: a) The cache energy efficiency per access from CPU; b) Reduction in the total amount of off-chip memory accesses.

2.2.4 Motivational Example

A filter cache structure [20] is used as the research target in this motivational example. To simplify the point of view, we assume that the total energy consumption of a cache system only consists of the dynamic energy for on-chip caches in the example. All miss penalties are set at 1 clock cycle to bring data into the on-chip cache.

Fig. 2.2 shows an example of three different memory configurations with a 6T-based L0 cache, a 10T-based L0 cache, and an SCM-based L0 cache system as (a), (b), and (c), respectively. The energy consumption per access and corresponding miss rate is also written in the figure. It should be mentioned all cache systems occupy the same on-chip area.

Since a 6T-based cache consumes more energy than low-voltage memories, this chapter replaces 6T with an NT memory in the L0 filter cache. When we replace 6T with a lower-area-density NT memory, the capacity is decreased, hence, the miss rate increases at the same time. If we can make good use of the non-linear relationship between capacity and miss rate, the energy overhead caused by the increase of the miss rate can be less than the energy reduction caused by implementing the NT-tolerant memory to the L0 cache. Here is a motivational example to explain the idea. The energy efficiency of the 10T-based L0 cache is 3 × better than that of the 6T-based L0 cache. At the same time, with the replacement of 6T with 10T, the L0 cache miss rate is only increased by 2% with the regress of the cache capacity. As a result, the 10T-based L0 cache has a larger energy reduction than the 6T-based one on the left in Fig. 2.3. As a result, the 10T-based L0 cache system achieves 52% less energy consumption than the 6T-based L0 cache. Following the

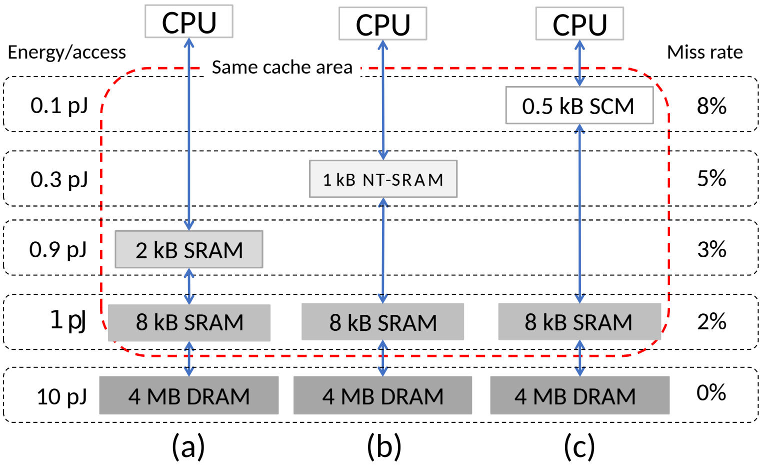


Figure 2.2: Three different cache architecture in the same on-chip area: (a) a 6T-based L0 cache, (b) a 10T-based L0 cache, (c) an SCM-based L0 cache.

same idea, using SCM instead of 10T, the miss rate is only increased by 3% but achieves 3 × more energy reduction than the 10T-based L0 cache. By replacing 10T with SCM, 28% total energy reduction is achieved compared with the 10T-based L0 cache.

2.3 Energy-Efficient Hybrid Cache System

In this section, we propose a hybrid cache structure to reduce the overall power consumption of the memory system under a given area constraint.

2.3.1 2-Level Hybrid Cache System

A hybrid 2-level on-chip cache system is proposed which is shown in Fig 2.4. Since the L0 cache has the highest access frequency and dominates the energy efficiency of the entire memory system, the L0 cache is built up of SCM which shows better energy efficiency than 10T SRAM. The L1 cache requires a larger capacity than the L0 cache to limit the energy-consuming off-chip memory access. Therefore, the L1 cache is built up of 6T-SRAM which shows better area efficiency than SCM. The rest of this chapter is to find an energy-efficient combination of SCM and 6T-SRAM capacities that minimizes the energy consumption of the entire cache system under various combinations of constraints.

Minimizing the L0 penalty enables a further energy reduction since it reduces the number of L1 read cycles and L0 write cycles when the L0 cache misses. Therefore, this chapter limits L0 cache penalty to 1 by an early restart technology [18], where a requested code is sent to the L0 cache and CPU simultaneously when the L0 cache misses.

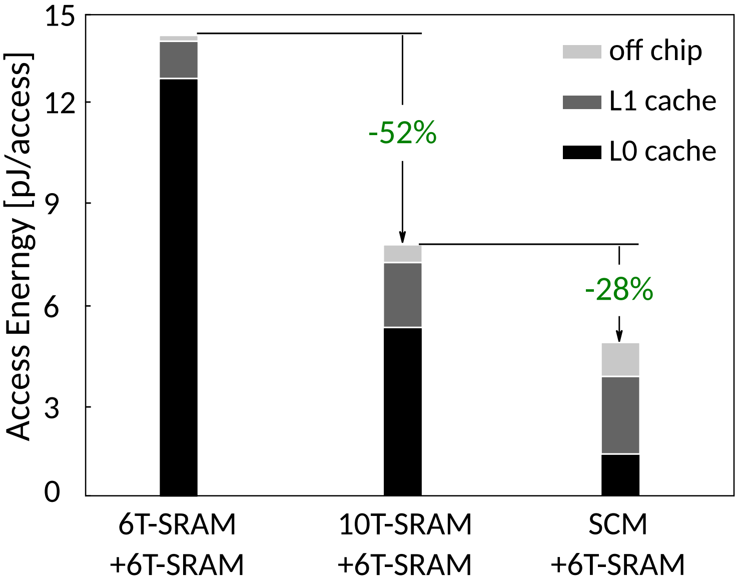


Figure 2.3: Access energy comparison of an example architecture.

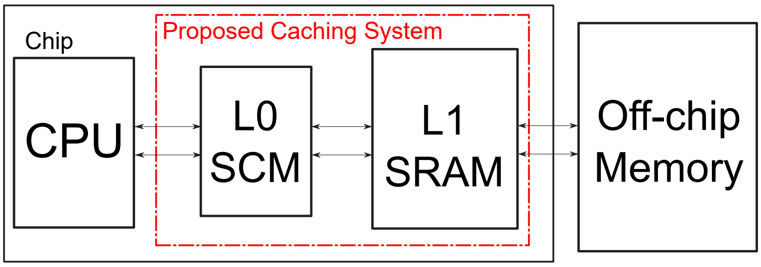


Figure 2.4: Proposed hybrid 2-level cache system consisting of an SCM-based L0 cache.

Since it only requires 1 cycle cache access, it has little penalty on energy consumption. The optimization of set associativity and block size is beyond the scope of this chapter. Therefore, this chapter specifies the caches discussed in Tab. 2.1. The associativity of the L0 cache and the L1 cache is 2-way. The line sizes of the L0 cache and the L1 cache are 16 words/line and 64 words/line, respectively.

Table 2.1: Cache setup for verification in RISC-V processor.

Cache location.	**Associativity**	**Line size**	**Miss penalty**
L0	2-way	16 words	1 clock
L1	2-way	64 words	64 clocks

2.3.2 Problem Definition

The energy consumption of a cache system can be modeled using the write and the read energy of a target memory. If a cache hit has occurred in the L0 cache, only read energy for one L0 cache access is consumed. If a cache miss has occurred for the L0 cache access, write energy in the L0 cache is consumed to write correct values back to the L0 cache. Moreover, the cache system accesses the higher-level cache (i.e. L1 cache) to request corresponding codes, which leads to extra energy overhead. In the same way, the energy consumption of the entire cache system can be iteratively defined in accordance with miss results of the L0/L1 cache and the off-chip memory access. The energy consumption per access of the whole cache system (E_{total}) can be expressed as follows:

$$E_{\text{total}} = E_{\text{read}} + E_{\text{write}} + E_{\text{leak}}, \tag{2.1}$$

where E_{read}, E_{write} and E_{leak} are energy consumption per access of the read energy, the write energy, and the leakage energy, respectively. E_{read} is expressed as follows:

$$E_{\text{read}} = R_{\text{L0}} + R_{\text{L1}} \cdot M_{\text{L0}} \cdot P_{\text{L0}} + R_{\text{off}} \cdot M_{\text{L0}} \cdot M_{\text{L1}} \cdot P_{\text{L1}}, \tag{2.2}$$

where R_{L0}, R_{L1}, R_{off}, M_{L0}, M_{L1}, P_{L0} and P_{L1} are L0 cache read energy per access, L1 cache read energy per access, off-chip memory read energy per access, L0 miss rate, L1 miss rate, L0 miss penalty and L1 miss penalty, respectively. The miss penalty is the number of extra clock cycles introduced by the cache miss. For example, if the L0 cache in Fig. 2.4 does not have instruction codes requested by CPU, a miss in the L0 cache occurs. When the L0 cache copies instruction codes from the higher level cache (i.e., L1 cache) due to the L0 cache miss, CPU suspends its pipeline, which leads to extra clock cycles introduced by the L0 cache miss. In the same way, the miss penalty for the L1 cache is defined as the number of clock cycles in which the L1 cache copies instruction codes from the off-chip memory due to the L1 cache miss.

E_{write} is expressed as follows:

$$E_{\text{write}} = W_{\text{L0}} \cdot M_{\text{L0}} \cdot P_{\text{L0}} + W_{\text{L1}} \cdot M_{\text{L0}} \cdot M_{\text{L1}} \cdot P_{\text{L1}}, \tag{2.3}$$

where W_{L0} and W_{L0} are L0 write energy per access and L1 write energy, respectively.

$$E_{\mathrm{leak}} = E_{\mathrm{leak}_{\mathrm{L0}}} + E_{\mathrm{leak}_{\mathrm{L1}}} + E_{\mathrm{leak}_{\mathrm{off}}}, \tag{2.4}$$

where $E_{\mathrm{leak}_{\mathrm{L0}}}$, $E_{\mathrm{leak}_{\mathrm{L1}}}$, $E_{\mathrm{leak}_{\mathrm{off}}}$ are L0 cache leakage energy consumption per access, L1 cache leakage energy consumption per access and off-chip memory leakage energy consumption per access.

The total access time is shown in Eq. (2.5):

$$T = (I + M_{\mathrm{L0}} \cdot P_{\mathrm{L0}} + M_{\mathrm{L1}} \cdot P_{\mathrm{L1}})/F \leq T_{\mathrm{constraint}}, \tag{2.5}$$

where I, F and $T_{\mathrm{constraint}}$ are the total number of instruction cycles, frequency and time constraint, respectively. Since the access time of NT memory is dominant in the proposed cache system, this chapter assumes that F is the same as the operating speed of the L0 cache.

We assume that the total area of the cache system can be expressed as Eq. (2.6).

$$A_{\mathrm{total}} = A_{\mathrm{L0}} + A_{\mathrm{L1}} \leq A_{\mathrm{constraint}}, \tag{2.6}$$

where A_{total}, A_{L0}, A_{L1} and $A_{\mathrm{constraint}}$ are the total area of the cache system, L0 cache area, L1 cache area and area constraint, respectively. This chapter assumes that the area of each cache is proportional to its capacity.

Under specific combination constraints, this chapter evaluates the energy consumption of the hybrid 2-level cache system with combined NT memory and 6T-SRAM. We utilize the analytical performance models Eq. (2.1) (2.5) (2.6) to find an energy-efficient combination of the L0 cache capacity, its V_{DD}, and the L1 cache capacity. In order to reduce the leakage energy consumption, the body bias (V_{BB}) is also tuned in this chapter.

2.4 Verification of Hybrid Cache for 65nm RISC-V Processors

This section evaluates a multi-level hybrid cache system implemented into a RISC-V processor [44].

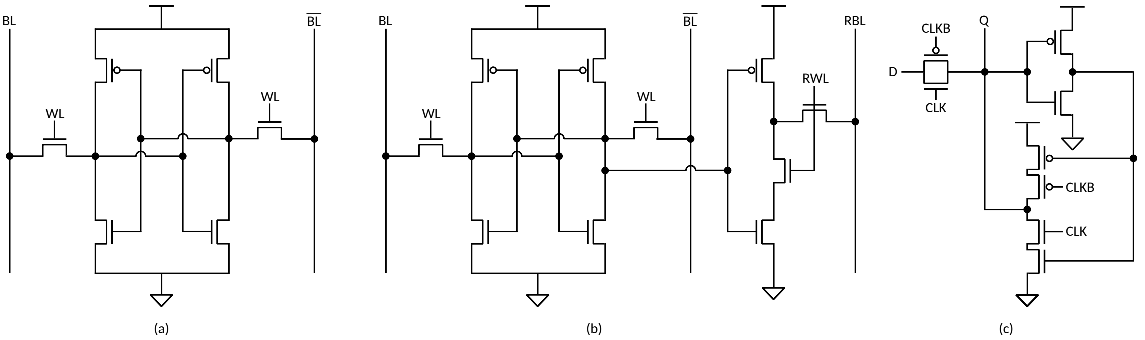


Figure 2.5: Candidate cells: (a) Differential 6T-SRAM, (b) 10T-SRAM and (c) Standard-Cell Memory.

2.4.1 Evaluation Setup

In order to verify the proposed hybrid cache system, a 2-level instruction hybrid cache memory is designed using a 65-nm process technology as a case study. This chapter selects a conventional 6T-SRAM, a 10T-SRAM, and an SCM-based L0 cache architecture.

Fig. 2.5 (a) shows a differential 6T-SRAM bit cell. As many papers point out, a stable operation in readout and write can no longer be guaranteed in the NTV region [39][43][45]. 10T-SRAM isolates read and write paths by providing two additional transistors in order to guarantee a stable operation in the NTV region [43]. 10T makes separate optimization available through isolating read and write operations. The specific method of isolation is to add two NMOS transistors in Fig.2.5 (b) dedicated to read operations, which are controlled by read word and read bit-line respectively to implement a single-ended read. This read circuity determines the read stability of the 10T cell. Since read and write operations are separated, we can independently improve the robustness of the write operation regardless of read operations to make it stable at the NT voltage region.

However, the robustness of read and write are accompanied by penalties of area. Since the read operation is single-ended, without differential sense amplifiers, 10T cells suffer from a larger delay. Therefore, the size of the NMOS transistor on the read path must be increased to mitigate delay penalties caused by the single-ended read operation. As a result, NT-tolerant 10T has at least 66% area penalty compared to conventional 6T [43]. This chapter considers differential 10T-SRAM as a representative of NT-tolerant SRAMs. Latch cells depicted in Fig. 2.5 (c) are implemented into SCM. Since SCM employs digital structures only, it can stably operate in the NTV region. At the same time, SCM consumes several larger area than SRAM due to its large readout logic.

Tab. 2.2 summarizes the comparison between three types of memory designs. When the supply voltage is near the threshold voltage, leakage powers of 10T and SCM are

Table 2.2: Comparison between 6T-SRAM, 10T-SRAM and SCM.

1 kB mem.	**6T-SRAM [15]**	**10T-SRAM [43]**	**SCM [42]**
Basis of results	65 nm model	65 nm AM	65 nm PLS
Supply voltage [V]	0.8	0.4	0.4
Freq.	1.3 GHz	586 kHz	30.6 MHz
Leak [pW/bit]	2.4	2.81	13
Dyn. Energy* [fJ/bit]	101	25.8	10.9
Cell area [μm^2/bit]	1.6	2.9	8.0
Energy × Area	161.6	74.8	87.2

*: Average energy consumption (read and write operations).
AM: ASIC measurement. PLS: Post-layout simulation.

greatly increased. The area overhead of 10T and SCM over 6T are 64% and 500%, respectively. The energy-area product is also listed in the table as a reference here.

To guarantee a stable signal transmission from the CPU core operating in the NTV region to the nominal-voltage 6T-SRAM, level shifters are implemented into corresponding data paths. This chapter considers the effect introduced by the level shifter. The evaluation is based on the level shifter proposed in [46]. The area of one level shifter is evaluated based on the actual layout in the 65-nm process technology. The result shows that the area of the level shifter is 6 μm^2. Since the supply voltage for the CPU core is tuned depending on a frequency constraint, this chapter evaluates the energy consumption and the delay of the level shifter with HSPICE under various voltage conditions. For example, if the input and the output voltages are 0.4 V and 0.8 V, respectively, the energy consumption and the delay of the level shifter are 5 fJ and 1.8 ns, respectively. The overhead introduced by level shifters is added to the performance models presented from Eq. (2.1) to Eq. (2.6). For example, the area overhead of level shifters implemented into the L1 cache memory is included in A_{L1} in (2.6).

As a summary, the following memories are used in the experiments:

- SCM: An SCM proposed in Ref. [42] is used as a representative of SCM. Area, energy, and operating speed are evaluated through a post-layout gate-level simulation in the commercial 65-nm process technology. The supply voltage is scaled from 0.4 V to 1.2 V. Body bias is scaled from −0.8 V to 0.4 V.

- 10T-SRAM: A 10T-SRAM model is used [43]. Supply voltage is scaled from 0.4 V to 1.0 V. Body bias is fixed at 0 V.

- 6T-SRAM: The low power SRAM model in CACTI Ver. 7.0 is used to estimate the performance of SRAMs [15]. Supply voltage is fixed at 0.8 V. Body bias is fixed at 0 V.

- Off-chip memory : Main memory mode in CACTI Ver. 7.0 is utilized [15].

Since the energy consumption of 6T-SRAM, 10T SRAM, and SCM are evaluated by the CACTI model, the ASIC measurement, and the post-layout simulation, respectively, the delay and the energy consumption caused by the wiring capacity of the data line and the selector are also included in the simulation of this chapter. In general, the performance of a memory depends on its capacity. However, Ref. [43] presents measurement results for a 32 kB capacity only. In order to estimate the delay and the energy consumption of 10T SRAM with various capacities, we utilize the capacity dependency of the 6T CACTI model [15]. For example, let us consider a case where the capacity of 6T-SRAM is reduced from 32 kB to 16 kB. Suppose the read energy determined by the 6T CACTI model is reduced by 28% through the capacity reduction. This chapter assumes that the read energy of 10T SRAM is also reduced by 28% if its capacity is reduced from 32 kB to 16 kB.

A code tracing log is firstly generated through a Register-Transfer-Level (RTL) simulation for a RISC-V processor running benchmark programs. Note that the processor has no cache memories in the log generation. Based on the code tracing log, the following cache activity is calculated assuming that the processor employs the corresponding cache system:

- The number of cache misses/hits,
- the number of cache read operations, and
- the number of cache write operations.

Combining the cache activity and the physical performance data for 6T-SRAM and NT-tolerant memories, this chapter evaluates the entire performance (i.e., delay, energy consumption, and area) of the target cache system. Note that this chapter does not consider the effect of control logic circuits, and a tag memory assuming that the performance of the NT-tolerant memory and the 6T-SRAM is a dominant source that determines the performance of the entire cache system.

2.4.2 Experimental Results

This chapter focuses on the environment such as sensor networks where several dedicated applications are executed on the chips. Therefore, this chapter utilizes several

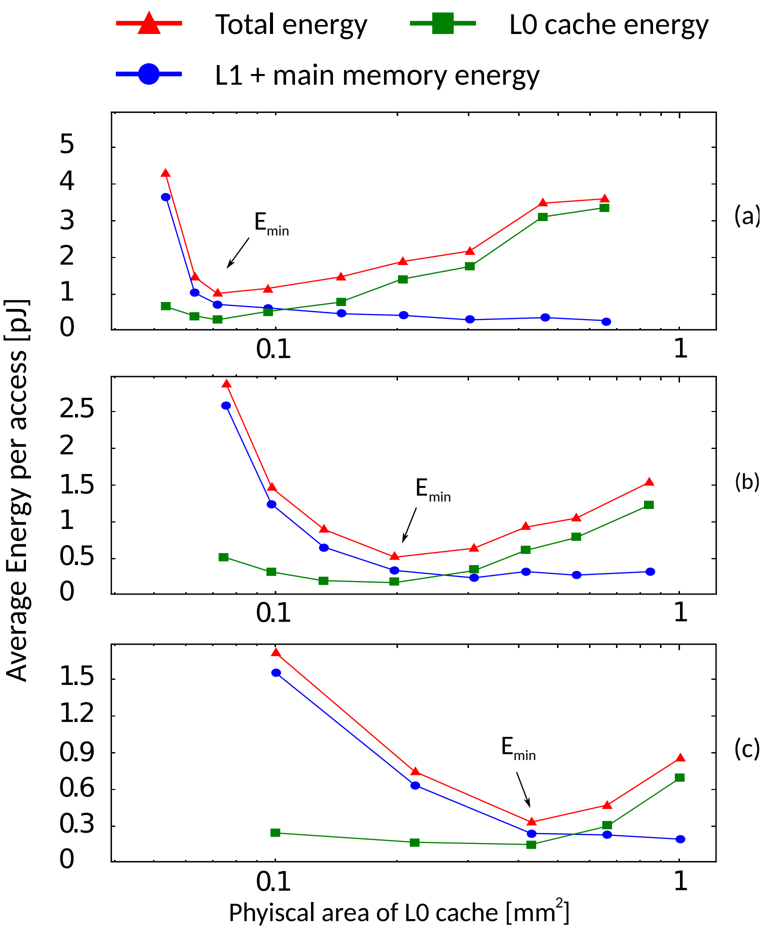


Figure 2.6: Three energy consumption per access of the memory system against the L0 cache capacity of DCT program: (a) Hybrid cache system with 6T-based L0 cache, (b) Hybrid cache system with 10T-based L0 cache, (c) Hybrid cache system with SCM-based L0 cache.

commonly used four typical programs for wireless communications, including Discrete Cosine Transform (DCT), Fast Fourier Transform (FFT), Secure Hash Algorithm 3 (KECCAK), Secure Hash Algorithm (SHA), and a mixed program which executes four programs sequentially, to evaluate the hybrid cache energy. This chapter refers to the mixed program hereafter as MIX. This chapter follows the methodology discussed in Section 2.3.2 to find an energy-efficient hybrid cache solution under different constraints. This chapter targets to find an energy-efficient hybrid cache system under a given area and frequency constraint. This chapter adjusts capacities of the L0 cache and the L1 cache to meet the area constraint. Therefore, by scaling V_{DD} of the NT-L0 cache, this chapter minimizes the total energy consumption of the cache system under the given time constraint.

The minimum energy point $E_{\min}$ can be found by considering the energy consumption of the current level cache and the next level cache. If we utilize more energy-efficient but low-density memory, the value of E_{min} should be smaller. At the same time, the required area to achieve E_{min} of the NT memory is also larger than E_{min} of 6T at the same time. This chapter evaluates the energy consumption in DCT to illustrate the claim in Fig. 2.6. The capacities of the L1 cache and the off-chip main memory are set as 16 kB and 4 MB, respectively. When the area of the L0 cache is greater than 0.1 mm^2, the miss rate tends to be convergent. Therefore, when using the 6T-based cache to execute the program, the capacity of the L0 cache reaches 4 kB when the area is less than 0.1mm^2. The reason is that the 6T's area density is the largest among the three candidates. The total energy consumption of the 6T-based cache system reaches E_{min} which has a value of 1.07 pJ/access, with the area of 0.1mm^2. The 6T-based L0 cache achieves minimum energy consumption at the same time. Moreover, the 10T-based cache achieves E_{min}, which is 0.64 pJ/access. At this moment, the 10T-based L0 cache consumes 0.2 mm^2 due to the relatively worse area density than the 6T cache.

The SCM-based hybrid cache system, which is more energy-efficient but low-density, achieves the smallest E_{min} among three candidates. However, where the area constraint is particularly tight, the SCM-based hybrid cache system can no longer reduce the energy consumption due to a sharply increased miss rate of the L0 cache. When we use the proposed SCM-based cache system, the SCM-based cache reaches E_{min} in the area of 0.4 mm^2 due to its low area density. However, since SCM has better energy efficiency than both 6T-SRAM and 10T-SRAM, E_{min} is reduced to 0.48 pJ/access, which indicates the total energy consumption of the SCM-based cache is 39% of the 6T-based cache system. Also, compared to two SRAM-based L0 caches, the proposed cache still exhibits smaller energy consumption when the area constraint is not less than 0.1 mm^2.

Figs. 2.7, 2.8 show the energy consumption under loose constraints (Area $<$ 0.4 mm^2, Freq. $>$ 1 MHz) of the proposed cache system when the processor executes benchmarks. The results are summarized in Tab. 2.4.2. Since the 10T cache has a higher area density than the 6T one, under the same area constraint, the 10T-based L0 capacity is twice as large as the SCM-based L0 one. However, because of SCM's better energy consumption than 10T SRAM, the total energy of the SCM-based hybrid cache system achieves 4.59 μJ while the 10T-based cache achieves the 9.26 μJ energy consumption. By applying a body biasing technique [47], the leakage energy can be reduced at the cost of degrading the operating speed of the cache system. Up to 1% energy consumption is reduced by the body biasing technique. This implies that the optimum body bias voltage is around the zero bias condition. The evaluation results for the rest programs are shown in Fig. 2.8. The total number of clock cycles required to execute the mixed program is around one million. In Fig. 2.8, the energy consumption is normalized by the energy of the 6T-based

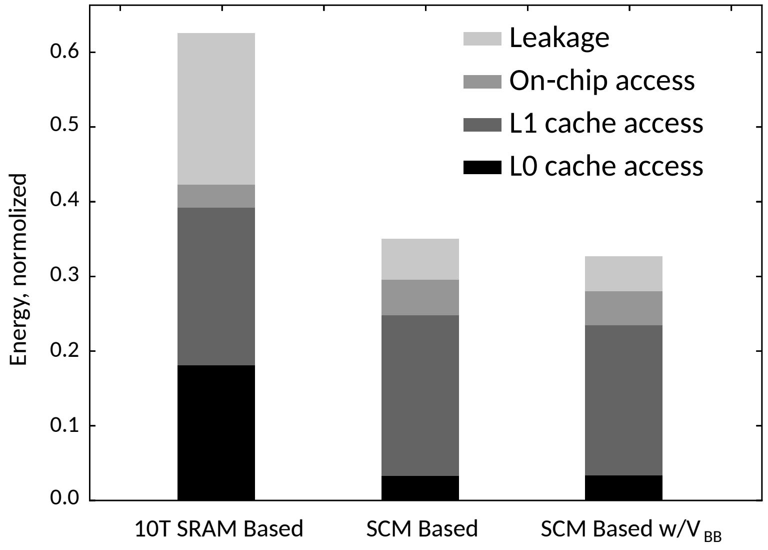


Figure 2.7: Energy consumption normalized by 6T-based cache system of MIX program under Area $<$ 0.4 mm^2, Freq. $>$ 1 MHz.

Table 2.3: Hybrid cache system parameters of the energy-efficient combination of MIX program under Area < 0.4 mm^2, Freq. > 1 MHz.

Cache Type	Energy	V_{DD}	C_fL0	C_{L1}	Freq.
10T-based	9.26 μJ	0.5 V	4 kB	8 kB	3.6 MHz
SCM-based	4.59 μJ	0.4 V	2 kB	8 kB	22 MHz
SCM-based w/ V_{BB}*	4.55 μJ	0.4 V	2 kB	8 kB	4.9 MHz

V_{DD} and V_{BB} are supply voltage and body bias for the L0 cache, respectively.
C_{L0} and C_{L1} are the capacities of two caches, respectively
*: V_{BB} is set as -0.7 V.

cache.

For the 2-level SRAM-based cache system, the miss rate is considerably low since the area efficiency of SRAM is high. The proposed hybrid system, on the other hand, exploits the high energy-efficiency of SCM and high area-efficiency of SRAM. As a result, if we target a 0.4 mm^2 area constraint, the proposed system achieves 51% better energy efficiency than the 2-level 10T-based cache system when processing the MIX program. The proposed cache can achieve 68% less energy consumption than the conventional 6T-based cache system when processing the MIX program.

The proposed SCM-based hybrid cache system achieves further energy reduction exploiting a trade-off between energy feature and area-density feature of the L0 cache. Utilizing the non-linear relationship between capacity and miss rate, this chapter replaces the conventional near-threshold 10T-based L0 cache with a more energy-efficient but low

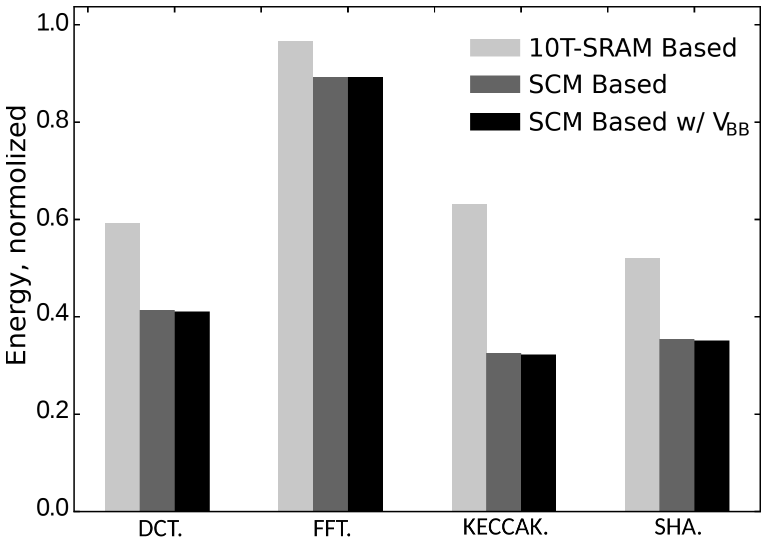


Figure 2.8: Energy consumption normalized by 6T-based cache system of DCT, FFT, KECCAK, SHA under Area < 0.4 mm^2, Freq. > 1 MHz.

area-density SCM one. As a result, more than half of energy consumption can be reduced.

2.4.3 Applicable Range Analysis

In the previous section, this chapter got the conclusion that the proposed cache system is more energy-efficient than the 10T-based cache system. However, if area constraints are too tight, the SCM-based hybrid cache suffers from the energy penalty introduced by its poor capacity. At the same time, the application such as IoT only requires a moderate operating speed. Analysis of the relationship between energy consumption and operating frequency can guide us in practical applications. Therefore, this section will further analyze the energy consumption of the proposed hybrid cache under different area and time constraints.

Figure 2.9 shows the energy consumptions of the 2-level caches with 10T-L0 and SCM-L0 as functions of area constraints. The energy consumption is normalized by the 2-level cache with SRAM-L0 under the same area constraint. Time constraints are not considered. Note that area constraints “0.1 mm^2” and “0.8 mm^2” in the X axis of Fig. 2.9, correspond to the area of SRAM 8 kB and SRAM 64 kB, respectively. Although the SCM area density is lower than that of the 6T-SRAM, the proposed SCM-based 2-level hybrid cache system has a large advantage due to its energy efficiency for speed-insensitive processing.

When an ultra low-energy oriented application is targeted, CPU requires near-threshold operation where 6T-SRAM cannot operate correctly due to its vulnerability against noise

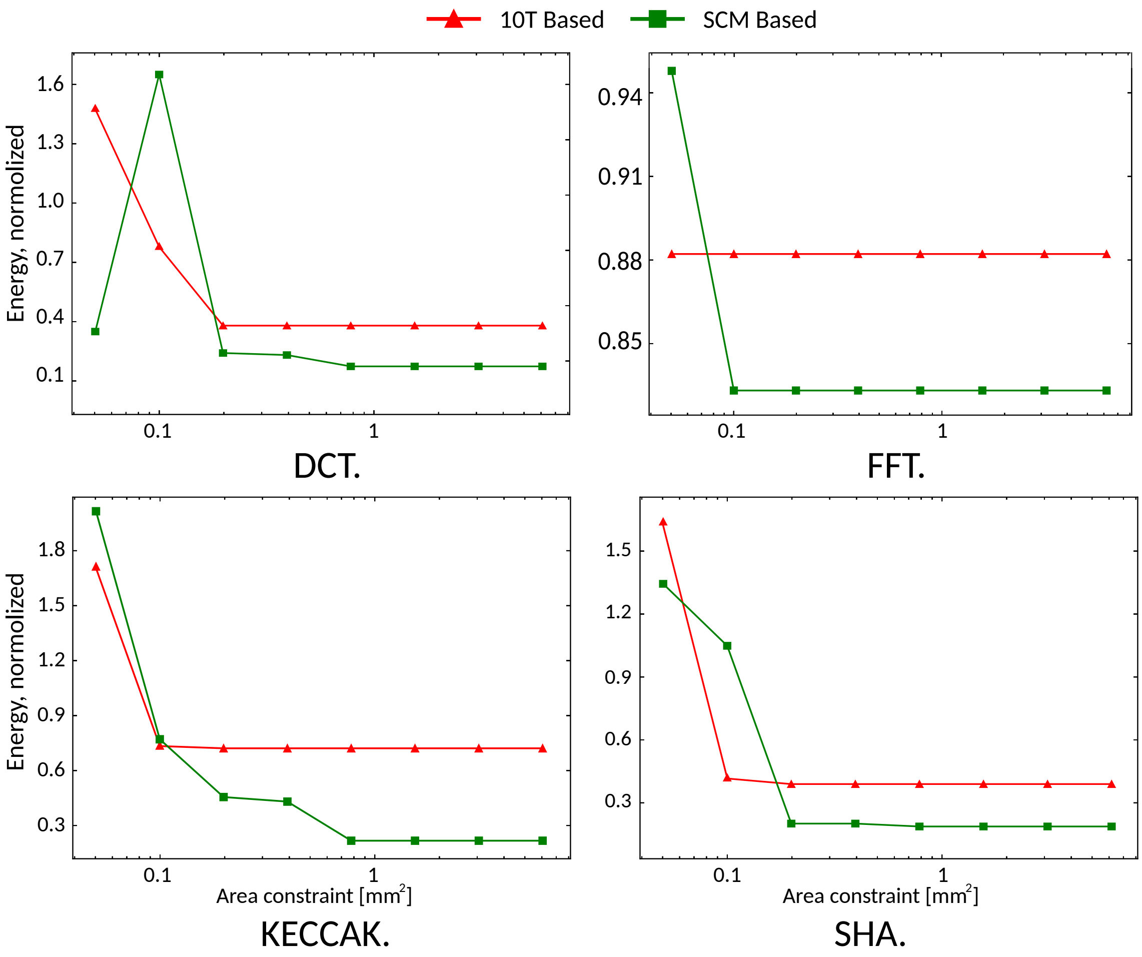
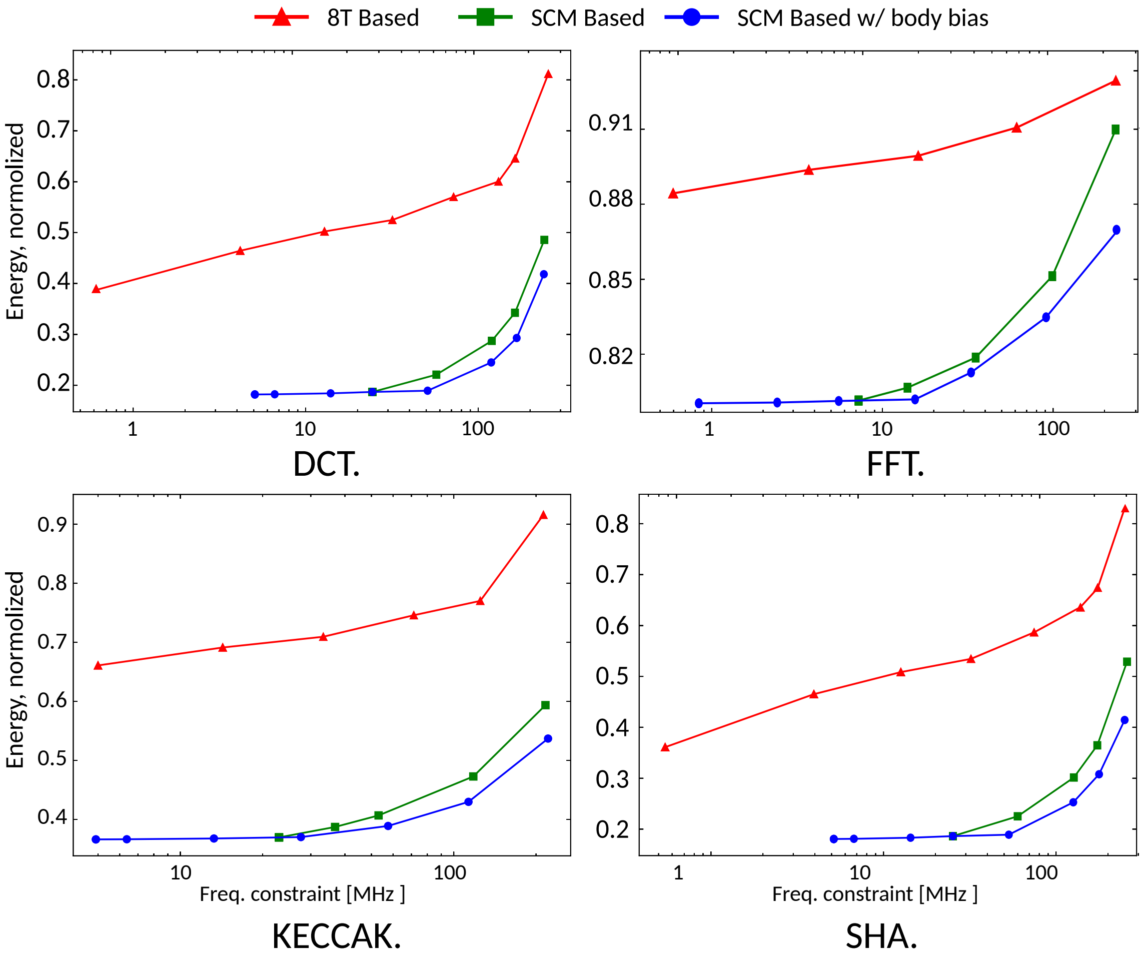


Figure 2.9: Normalized energy comparisons among three 2-level caches against 6T-based cache system under a loose freq. constraint towards the mixed program.

and process variability. Therefore, with a 6T-only configuration for the cache, we cannot lower the energy consumption of CPU sufficiently even if the performance required for a target application is very low. Compared to a 10T-based hybrid cache design, although SCM has a lower area density, in the cases when area constraints are not lower than 0.1 mm^2 from the simulation results, the SCM-based 2-level cache has an energy reduction due to energy feature advantages.

Fig. 2.10 shows evaluation results under a loose area constraint. Note that frequency constraints are tuned unlike Fig. 2.9. The X axis represents a frequency constraint. The Y axis is the normalized energy consumption against the 2-level hybrid cache system with the 6T-based L0 cache under the same area constraint, which is the same as Fig. 2.9. Area constraints are not considered here. When frequency constraint is tightened, the NT-tolerant cache system must raise the L0 cache V_{DD}, which leads to the energy overhead. When the target frequency is higher than 100 MHz, the entire cache system will not work



DCT.

FFT.

KECCAK.

SHA.

Figure 2.10: Normalized energy comparisons of 2-level hybrid caches against 6T-based cache system under a loose area constraint towards the mixed program.

properly because NT-L0 cannot meet the frequency requirement. However, if we relax the timing constraint, the proposed cache exhibits 64% better energy efficiency than the 2-level 10T-based cache at the best case shown in Fig. 2.10.

The reduction in energy consumption introduced by body bias optimization is no more than 1% when the target frequency is less than tens of MHz. As described before, this is because the optimum body bias condition for the SCM is around the zero bias condition. However, if the frequency constraint becomes more than tens of MHz, the situation becomes different. In order to boost up the operating speed of the SCM, the supply voltage is typically boosted up, which leads to an increase of the dynamic energy consumption. By applying forward body biasing, the supply voltage can be reduced without violating the frequency constraint. As a result, the energy consumption of the proposed SCM-based cache can be reduced by up to 26% by applying forward body biasing.

In summary, when the area constraint is not less than 0.1mm^2 and the frequency constraint is not more than 100 MHz, SCM-based L0 cache achieves total energy reduction by a trade-off between energy efficiency improvement but area feature reduction.

The essence of trade-off is to exploit the nonlinear relationship between capacity and miss rate. When the area constraint is loose, the cache capacity is large enough hence the descrease of cache capacity will not severely increase the miss rate. In such a situation, energy reduction is expected with the replacement of a more energy-efficient but low-capacity SCM-based cache. However, when the area constraint is too small, the corresponding cache capacity will be small. Therefore, a slight cache capacity reduction will also lead to a rapid increase in the cache miss rate, making this trade-off no longer energy-efficient.

2.5 Summary

Based on the non-linear relation between cache miss rate and capacity, a case study is discussed to show SCM-based design has an energy advantage than 10T-based design. SCM has a low processing speed, a low area density but an inherent energy efficiency. Utilizing SCM as the L0 filter cache, the 2-level hybrid cache system exhibits 68% better energy consumption than a conventional 6T-based cache system in the simulation on average. When an ultra-low-power oriented application is targeted, the CPU requires near-threshold operation where 6T cannot operate correctly due to its vulnerability against noise and process variability. If we utilize 10T for near-threshold operation, due to the SRAM read operation by a strong sense amplifier, it is still not energy-efficient enough. In contrast, if the CPU is under a near-threshold operation, the SCM operates very well with the same supply voltage as the CPU logic. Based on this work, this chapter found that in an ULP working environment, the energy efficiency of a lower-level cache is far more important.

Chapter 3

DNN Accelerator Designs for Large-Scale Processing with Regular Access Pattern

3.1 Introduction

In recent years, the prediction accuracy of Deep Neural Network (DNN) has been steadily increasing with the improvement of network structure [48]. As a result, DNN has been widely used in all aspects of life such as image processing and language modeling. Therefore, many endpoint terminals integrate Application-Specific Integrated Circuits (ASICs) to perform local real-time inference [49–51]. However, the high-precision prediction is often accompanied by an increase in computational complexity [52, 53].

In DNN processing, most operations are matrix operations, such as convolutions and matrix multiplications [2, 54]. In matrix operations, expensive data movement often dominates the energy consumption of an entire computing architecture [2]. Ref. [4] proposes a key property called *data reuse* to expose the number of data movements in DNN processing, which describes the number of accesses to the same data during the processing. Based on data reuse, previous works [5–9] build evaluation models to estimate hardware performance.

Although the data reuse describes the overall attribute of DNN processing, it is not enough as an evaluation indicator to describe the quantitative difference among the cost of data movement in different processing dataflows. Because of this, Ref. [55] exhausts all possible processing dataflows and corresponding hardware implementations to explore the best one with the least cost of the data movement. On the other hand, the number of data movements is also important for the cost of data movement. However, the cost of one data movement is another important factor, which is difficult to be described by data reuse.

To point out the importance of the movement cost, this chapter explores the hardware property that determines the required memory capacity which is the key to the cost of one data movement. This chapter makes the following contributions:

1. This chapter first explores properties in a processing dataflow that can significantly affect the required number of memory accesses and the required memory capacities in a hardware implementation.

2. By improving the reuse strategy, this chapter proposes a convolution processing dataflow to minimize the number of on-chip memory accesses and the memory capacity.

3. Based on the properties, this chapter proposes quantitative metrics to evaluate the cost of data movement in a specialized processing dataflow. Given physical constraints such as the limitation of memory bandwidth, and the number of Arithmetic Logic Units (ALUs), the proposed metrics can estimate energy, throughput, and area of a specialized processing dataflow without hardware implementation.

4. To verify the effectiveness and flexibility of the proposed metrics, this chapter makes comparisons between the proposed architecture and a state-of-the-art DNN accelerator [55] for processing Alexnet [56].

The rest of this chapter is organized as follows. Section 3.2 discusses the motivation and the design space for DNN processing. Section 3.3 presents a dataflow that fully utilizes resource reuse in convolutions (CONVs). Section 3.4 shows implementation examples of the proposed dataflow. Section 3.5 presents quantified metrics to evaluate the energy, the throughput, and the area for a hardware implementation of a processing dataflow. Section 3.6 introduces two case studies for convolution and matrix multiplication which are common in DNN processing. Section 3.7 discusses the simulation results of the hardware realization to testify the accuracy of the proposed metrics. Section 3.8 concludes this chapter.

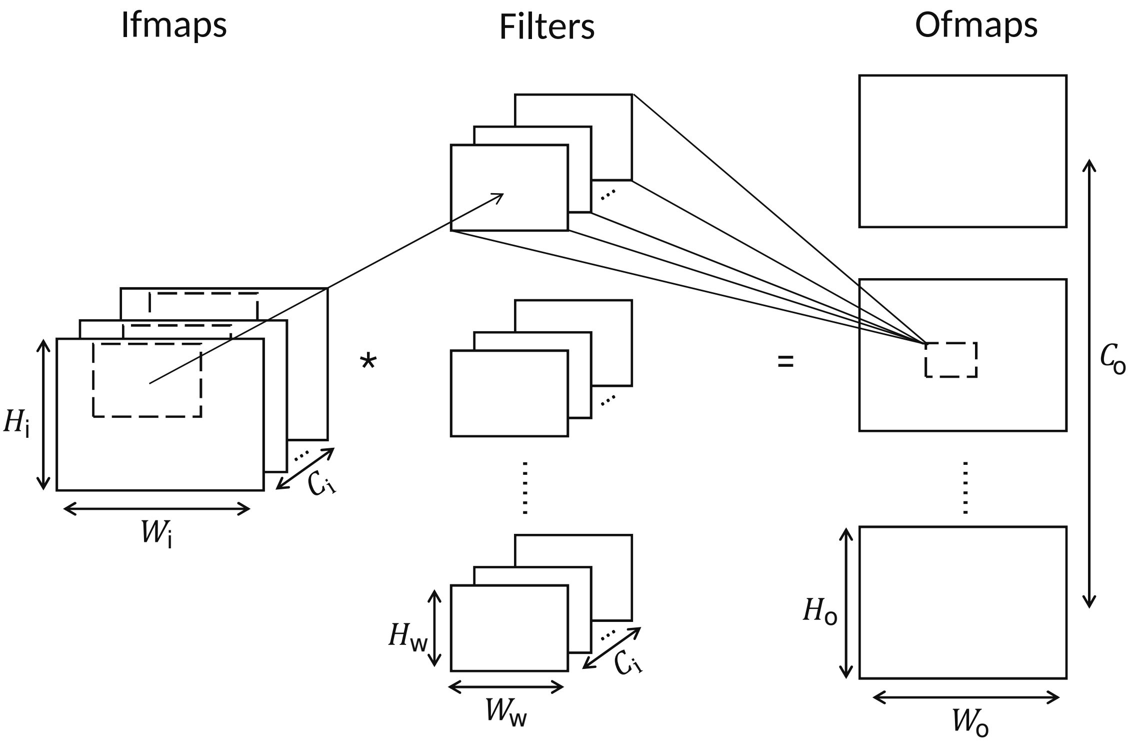


Figure 3.1: One batch of high dimensional convolutions in DNN: $C_i \times C_o \times H_w \times W_w \times H_o \times W_o$ multiplications are required in total.

3.2 Background

3.2.1 DNN Processing

DNN generates classification results by performing feature extraction from raw input data. DNN improves model accuracy by a very deep hierarchy of layers in the network. Among DNN, Convolution Neural Network (CNN) is most commonly applied to a wide range of applications including image recognition, recommendation systems, natural language processing, etc. As primary layers in CNN, CONV layers use a group of weights to extract high-level features which are called output feature maps (ofmaps) from input feature maps (ifmaps). Fig. 3.1 introduces a typical convolution in CNN. A traditional CONV layer in CNNs often has multi-channel 2-dimensional (2D) ifmaps and multi-channel 2D weights to enhance prediction accuracy [57]. Given the convolution shapes in Tab. 3.1, a CONV processing with multiple batches can be expressed as:

Table 3.1: DNN shapes.

Parameters	Description
N	Ifmap Batch Size
$H_\mathrm{i}/W_\mathrm{i}$	Ifmap Height / Width
$H_\mathrm{w}/W_\mathrm{w}$	Weight Height / Width
$H_\mathrm{o}/W_\mathrm{o}$	Ofmap Height / Width
C_i	Ifmap Channel Size
C_o	Ofmap Channel Size
U	Stride Size

$$\begin{aligned}\mathbf{O}[z][u][x][y] = \sum_{k=1}^{C_\mathrm{i}} \sum_{i=1}^{W_\mathrm{w}} \sum_{j=1}^{H_\mathrm{w}} \Big(&\mathbf{I}[z][k][Ux+i][Uy+j] \\ &\times \mathbf{W}[u][k][i][j]\Big) + \mathbf{B}[u], \qquad (3.1) \\ 1 \le z \le N,\ 1 \le u \le C_\mathrm{o},\ 1 \le x \le W_\mathrm{o},\ 1 \le y \le H_\mathrm{o}&, \\ H_\mathrm{o} = (H_\mathrm{i} - H_\mathrm{w} + U)/U,\ W_\mathrm{o} = (W_\mathrm{i} - W_\mathrm{w} + U)/U&,\end{aligned}$$

where **O**, **B**, **W** and **I** indicate the matrices of ofmaps, biases, weights and ifmaps, respectively [2]. Eq. (3.1) can describe most primitive operations in DNN processing such as the depthwise separable convolution [53] and the fully-connected layer (FC) [48]. Therefore, the evaluation target of this chapter is mainly focused on the processing based on Eq. (3.1).

Other primitive operations in DNN can also be represented by Eq. (3.1). In order to convert the high-level extraction features from CONV layers into the classification results, Fully-Connected (FC) layer is used between CONV layer and the output layer [48]. Recent CNN has developed depthwise separable convolution [53] to diminish CONV channels to save energy. Pointwise convolution acts as the operations with $H_\mathrm{w} = W_\mathrm{w} = 1$ with Eq. (3.1). In mathematics, the operations in FC layers and Multilayer Perceptron (MLP) [48] can be represented by Eq. (3.1) with $H_\mathrm{i} = W_\mathrm{i} = H_\mathrm{w} = W_\mathrm{w} = 1$. The depthwise convolution consists of the 2D CONV processing with multiple ifmap channels and the same number of ofmap channels. In this case, the accumulation pattern is different from naive convolution operations. We thus need to utilize another dimension to describe the channel in depthwise

Table 3.2: Relations between CONV and other primitive operations in DNN.

Processing	Description
Pointwise Convolution	$H_\mathrm{w} = W_\mathrm{w} = 1$ with Eq. (3.1)
Depthwise convolution	Eq. (3.2)
FC (Matrix Multiplication)	$H_\mathrm{i} = W_\mathrm{i} = H_\mathrm{w} = W_\mathrm{w} = 1$ with Eq. (3.1)

convolution which will be represented by the following equation.

$$\begin{aligned}\mathbf{O}[z][d][x][y] = \mathbf{B}[z][d] + \sum_{\mathrm{i}=1}^{W_\mathrm{w}} \sum_{\mathrm{j}=1}^{H_\mathrm{w}} &\mathbf{I}[z][d][Ux+i][Uy+j] \\ &\times \mathbf{W}[d][i][j], \qquad (3.2)\\ 1 \le z \le N,\ 1 \le d \le D,\ 1 \le x \le W_\mathrm{o},\ 1 \le y \le H_\mathrm{o},& \\ H_\mathrm{o} = (H_\mathrm{i} - H_\mathrm{w} + U)/U,\ W_\mathrm{o} = (W_\mathrm{i} - W_\mathrm{w} + U)/U,&\end{aligned}$$

The relationship is summarized in Tab. 3.2. Since stride U does not affect the key characteristics of convolution, we default all U in DNN processing to 1 in the rest of this chapter.

3.2.2 Data Reuse in DNN

Ref. [4] proposes a key property called *data reuse* to expose attributes of data movement in DNN processing, which is to describe the number of accesses to the same data during the processing. Previous works [24–27, 30] exploit data reuse to design processing dataflows, which have achieved great success to reduce the number of data movement in DNN processing. A processing dataflow describes the sequence of data movement in the memory hierarchy and the sequence of executions in ALUs. Once the dataflow is determined, the required number of memory accesses and required memory capacity for each memory component would be determined.

The scale of DNN processing tasks and the shape of DNN processing are quite diverse. Furthermore, available hardware resources vary dramatically depending on the processing environment from end-point devices to servers. An exhaustive approach to explore the best dataflow with the least cost of the data movement from all possible dataflows makes it difficult to design accelerators in everchanging environments.

To solve this issue, this chapter no longer focuses on overall properties for a total processing task but properties for the processing task loaded in a processing hardware. The number of ALUs in a hardware accelerator is often much smaller than the required

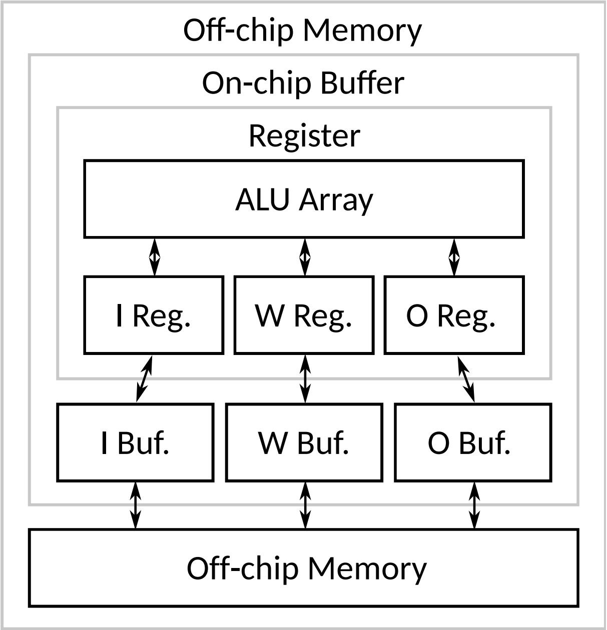


Figure 3.2: A target acceleration architecture:

operations of a total DNN processing task, which means that ALUs are hard to exploit all data reuse in the whole processing task. This makes it difficult for an overall attribute such as data reuse to evaluate quantitative differences among accelerators based on different processing dataflows. Therefore, this chapter focuses on the processing task loaded in the hardware to evaluate the hardware performance.

As the second part, this chapter explores properties in a processing dataflow that can significantly affect the cost of data movement in a hardware implementation. The processing dataflow divides the whole processing task into smaller tasks to fit the scale of the hardware and processes the tasks one by one [58]. As a result, it is required to consider the properties in the tasks to infer the cost of data movement.

3.2.3 Subtasks in DNN Processing

In order to compare different dataflows in DNNs, we further clarify essential differences among processing dataflows. The number of operations for most existing DNN computing tasks ranges from millions [59] to billions [60]. It is difficult for acceleration hardware to realize fully parallel processing in such a scale. Therefore, we have to face a situation where the number of operations that can be processed within an accelerator is much smaller than the total number of operations. Therefore, the architecture of the target accelerator in this chapter is introduced in Fig. 3.2. Hardware components in the architecture consist of an

ALU array, registers, on-chip buffers, and off-chip memories. The hardware architecture may have a different Processing Element (PE) structure such as a systolic ALU array or a spatial ALU array. An ALU may include a multiplier or multiplier-and-accumulation (MAC) unit. The ALU array with the registers in Fig. 3.2 corresponds to a single PE or multiple PEs working in parallel. Each memory hierarchy level consists of memories to store weights, ifmaps and ofmaps, respectively. The data transmission network between hardware components may also have different communication bandwidths and different data transmission techniques such as unicasting, multicasting, or broadcasting.

Under the situation of limited processing resources, it is required to decompose an entire processing task into *subtasks* to adapt the number of multipliers in each PE, which is the same as *tiling level* in a PE defined in Ref. [7]. Each tiling level has a loop corresponding to each dimension in the original processing task. The tiling level in a PE contains the operation space and the associated dataspaces within registers in a PE [7]. One PE usually consists of multiple multipliers and registers. A subtask in a PE refers to the dimensions of processing that can be processed within data in one PE. Each subtask can be completed within a PE once data loaded into the registers inside. In DNN processing, a subtask refers to the processing task in the dimensions of N, C_{i}, C_{o}, H_{o}, H_{w}, W_{o}, W_{w}, or the multi dimensions such as $(H_{\mathrm{o}}, H_{\mathrm{w}})$ and $(H_{\mathrm{w}}, W_{\mathrm{w}})$.

Each subtask corresponds to a specific processing hardware implementation. It is the subtask in the processing dataflow that determines the hardware performance. Therefore, when we compare different processing dataflows, we should concentrate on the properties of the subtask.

We use ALU subtasks, register subtasks, buffer subtasks, and memory subtasks to represent processing tasks in ALUs, registers, on-chip buffer, and off-chip memory, respectively. More specifically, we define an ALU subtask as the processing task that can be processed in parallel in ALUs. We use a register subtask to indicate the processing task that can be processed once the registers are fully filled until the next on-chip buffer access is required. Similarly, a buffer subtask and a memory subtask can be defined. This chapter describes properties for subtasks in a processing dataflow that affect the hardware.

Based on the properties for subtasks, this chapter proposes quantitative metrics to evaluate the cost of data movement in a specialized processing dataflow as the second part. Given physical constraints such as the limitation of memory bandwidth, memory capacity, and the number of ALUs, the proposed metrics can estimate energy, throughput, and area of a specialized processing dataflow without hardware implementation. To verify the effectiveness and flexibility of the proposed metrics, we make comparisons between the proposed architecture with state-of-the-art DNN accelerators [24, 55] for the processing

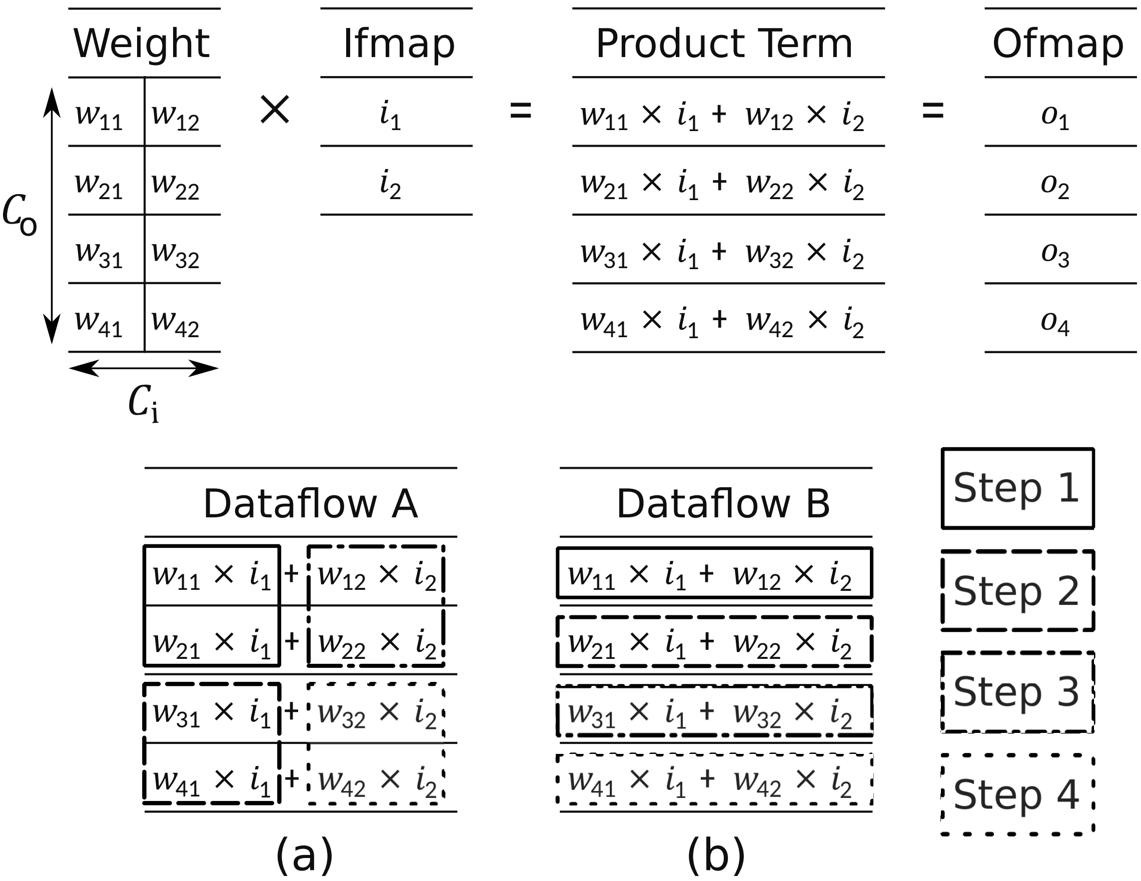

Weight
Ifmap
Product Term
Ofmap
C_o
C_i
w_{11} w_{12}
w_{21} w_{22}
w_{31} w_{32}
w_{41} w_{42}
×
i_1
i_2
=
$w_{11} \times i_1 + w_{12} \times i_2$
$w_{21} \times i_1 + w_{22} \times i_2$
$w_{31} \times i_1 + w_{32} \times i_2$
$w_{41} \times i_1 + w_{42} \times i_2$
=
o_1
o_2
o_3
o_4
Dataflow A
$w_{11} \times i_1$ + $w_{12} \times i_2$
$w_{21} \times i_1$ + $w_{22} \times i_2$
$w_{31} \times i_1$ + $w_{32} \times i_2$
$w_{41} \times i_1$ + $w_{42} \times i_2$
(a)
Dataflow B
$w_{11} \times i_1 + w_{12} \times i_2$
$w_{21} \times i_1 + w_{22} \times i_2$
$w_{31} \times i_1 + w_{32} \times i_2$
$w_{41} \times i_1 + w_{42} \times i_2$
(b)
Step 1
Step 2
Step 3
Step 4

Figure 3.3: Matrix Multiplication with $C_\mathrm{i} = 2$ and $C_\mathrm{o} = 4$, $N = H_\mathrm{w} = W_\mathrm{w} = H_\mathrm{i} = W_\mathrm{i} = H_\mathrm{o} = W_\mathrm{o} = 1$. Two possible processing dataflows with 2 multipliers are introduced. Dataflow A is the parallel processing to share ifmap pixels in each step. Dataflow B is the parallel processing to share ofmap pixels in each step.

in Alexnet [56].

3.2.4 Motivational Example

In this sub-section, we review a matrix multiplication shown in Fig. 3.3. Each ofmap pixel (o_1, o_2, o_3, and o_4) is the sum of two product terms. Based on the definition in Ref. [27], the data reuse of weights, ifmaps, and ofmaps are 1, 4, and 2, respectively.

Suppose we have 2 multipliers available, two processing dataflows A and B are possible as shown in Fig. 3.3 (a) and (b). Dataflow A utilizes 2 multipliers to generate product terms from the same ifmap pixel at the same time. Dataflow A reduces the number of accesses to an ifmap pixel, thereby exploiting data reuse of ifmaps. Similarly, Dataflow B calculates two product terms in parallel for one same ofmap pixel. Each step completes multiply and accumulates for each ofmap. Dataflow B exploits data reuse of ofmaps. However, since

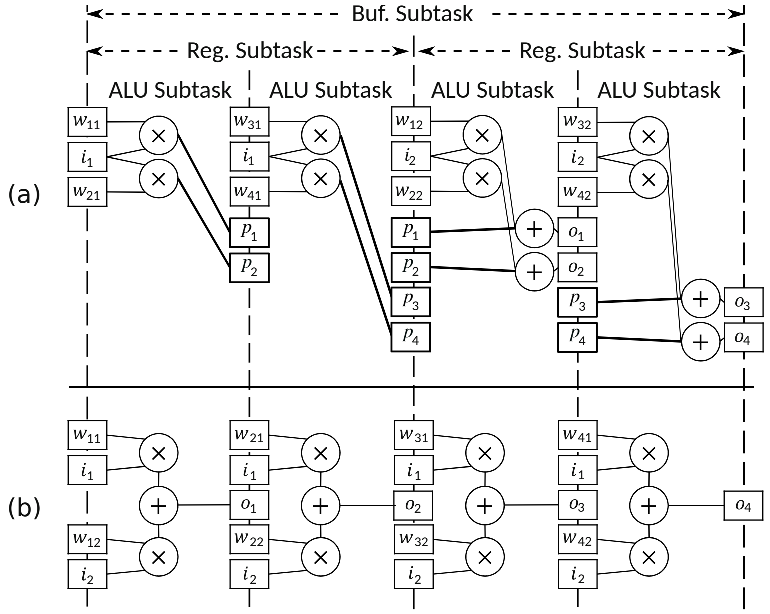


Figure 3.4: Hardware Implementations based on two schedulings of dataflow graphs. All ifmap pixels and weights have been loaded into on-chip buffers. Each step refers to one set of parallel processing according to two available multipliers. Each rectangle on the borderline between steps represents a register that can hold one data. The architecture in (a) refers to a hardware implementation of Dataflow A in Fig. 3.3. The architecture in (b) refers to a hardware implementation of Dataflow B in Fig. 3.3. Registers for product terms are reused until they are accumulated into final ofmap pixels.

both dataflows exploit data reuse with 2 multipliers, it is difficult to directly point out which processing dataflow has smaller energy consumption in the hardware implementation or smaller required memory capacity.

If we implement the two dataflows separately into hardware, the required hardware resources and energy consumption can be inferred from the scheduling of the dataflow graphs. Fig. 3.4 introduces hardware implementations based on the dataflows in Fig. 3.3. Each step refers to one set of parallel processing by two available multipliers. Each rectangle on the borderline separating steps represents a register that can hold one data. For initialization, all required data have been loaded into on-chip buffers. We assume that ALUs only access registers and the on-chip buffer cannot directly be accessed by ALUs. We omit accesses to on-chip buffers in Fig. 3.4 since both architectures are designed to minimize the number of accesses to the on-chip buffer so that the number of accesses to the on-chip buffer is the same for both implementations.

In order to collect the number of register accesses and the register capacity, we focus on the properties in the processing task of each step. The architecture in Fig. 3.4 (a) based on Dataflow A shares the access to one ifmap pixel thus reducing register accesses of ifmaps. Although the data reuse of ifmaps is 4 in the total processing task, one ifmap pixel can only be utilized for two multiplications per register access. The architecture (a) requires additional registers to store product terms, and access to product-term registers also brings additional energy overhead. Since the parallel processing of Dataflow B is for the same ofmap pixel, all product terms generated can be accumulated into the ofmap pixel immediately. Although the architecture (b) requires two separate accesses to each ifmap pixel, it requires fewer registers than the architecture (a) since no product-term register is needed. As a result, the architecture based on Dataflow A has more product-term registers and suffers from the corresponding energy overhead.

This chapter aims to analytically evaluate the cost of data movement in the overall hardware implementation based on a specialized processing dataflow without the hardware implementation nor the loop-based performance evaluation [6–8]. The key idea is to focus on the cost of data movement in each hardware component, which consists of both the number of data movement and the cost of one data movement. In the motivational example, both architectures contain three hardware components as buffers, registers, and ALUs. Through the properties of each hardware component, we evaluate the number of memory accesses and the memory capacity to estimate the cost of data movement of each component in the hardware implementation.

3.2.5 Design Space Exploration for DNN Accelerators

In order to cover the whole design space in DNN dataflow processing, we focus on the hardware architecture shown in Fig. 3.2. The number of operations for most existing DNN tasks may exceed billions [60]. In this chapter, the number of operations is defined as the number of product terms generated from multiplications. Therefore, we have to face a situation where the number of operations that can be processed within a hardware is much smaller than the total number of operations. With limited hardware resources, the decomposition of the entire processing task comes to be a must.

Each subtask describes a small loop in the original processing task. Fig. 3.3 introduces subtasks for both architectures of Fig. 3.4. An ALU subtask in both dataflows aims to generate 2 product terms in one step. A register subtask can be scaled from one ALU subtask to the total processing task for Performance-Power-Area trade-off. To avoid frequent buffer accesses caused by generated product terms, a register subtask is better not to be smaller than 2 ALU subtasks in Dataflow A. If the scale of a register subtask is set too large, there will be a register capacity overhead. As a result, we assume that a register subtask consists of 2 ALU subtasks in this example. A register subtask in Dataflow B is also defined as a processing task in 2 steps for a fair comparison. A buffer subtask is the total processing task.

3.3 DNN Dataflow to Reduce the Cost of Data Movement

3.3.1 Properties for the Cost of Data Movement in DNN Processing

In DNN processing, the cost of data movement dominates the throughput and the energy consumption of DNN accelerators [2]. Fortunately, for DNN processing, especially for matrix operations, the data access pattern is usually fixed. Therefore, we can find quantitative properties to describe the cost of data movement in the whole DNN computations. If we describe properties to evaluate the cost of data movement determined by each subtask in a processing dataflow, we can evaluate the overall acceleration hardware. In this sub-section, we describe properties in each subtask that determine the cost of data movement.

The cost of data movement in memories is determined by the memory capacity and the number of memory accesses. In order to evaluate the memory capacity and the number of memory accesses from the processing dataflow, we describe properties that determine the number of operations that use the same data in subtasks and the number of product terms that use the same memory space. In this chapter, the number of operations refers specifically to the number of multiplications that generate product terms.

We refer the number of operations for one single data in each subtask to *local data reuse* (LDR). The concept of LDR is almost the same as the data reuse which is proposed in Ref. [27]. The difference is that the data reuse aims to describe the overall processing task while LDR only focuses on a subtask. More specifically, LDR of an ifmap pixel in an ALU subtask is defined as the number of operations that one ifmap pixel generates in an ALU subtask. LDR of the weight is defined similarly. LDR of an ofmap pixel in an ALU subtask is defined as the number of product terms for the same ofmap pixel in an ALU subtask. LDR of an ifmap pixel in a register subtask is defined as the number of operations that one ifmap pixel generates in a register subtask. LDR of the weight is defined similarly. LDR of an ofmap pixel in a register subtask is defined as the number of product terms for the same ofmap pixel in a register subtask. Local data reuse aims to describe the property that affects the number of required data in one subtask, which determines the required memory capacity.

Basically, the number of memory accesses can also be evaluated based on LDR. We are able to evaluate the number of register accesses combining the LDR in an ALU subtask

Table 3.3: Hardware properties of two architectures in Fig. 3.4

Subtask	Num. of Ops	Property	Dataflow A			Dataflow B		
			W	I	O	W	I	O
ALU	2	LDR	1	2	1	1	1	2
Register	4	LDR	1	4	1	1	2	2
		LSR	1	4	2	1	4	2
Buffer	8	LDR	1	4	2	1	4	2
Reg. Capacity			4	1	4	4	2	2
Reg. Access			8	4	8	8	8	4
Buf. Capacity			8	2	4	8	2	4
Buf. Access			8	2	4	8	2	4

and the total number of operations. However, in a well-designed processing dataflow, the total number of accesses to on-chip buffers can be further reduced by keeping some data within registers to be shared among several register subtasks. The number of accesses to off-chip memories can also be reduced similarly. Therefore, it is necessary to introduce the property describing data reuse among subtasks to evaluate memory accesses. We define the number of operations for one same data until the data is removed from the current memory as *local space reuse* (LSR). LSR of an ifmap pixel in a register subtask is defined as the number of operations that one ifmap pixel generates until the ifmap pixel is removed from the current register. LSR of a weight and an ofmap pixel is defined similarly. Local space reuse aims to describe the property of data utilization among multiple subtasks within the memory.

Hardware properties for both Dataflows A and B are summarized in Tab. 3.3. One ALU subtask in Dataflow A generates 2 product terms. One ifmap pixel is shared by two multiplications in an ALU subtask. The data sharing does not exist in weights nor ofmap pixels in an ALU subtask. As a result, LDR of weights, ifmaps, and ofmaps in each ALU subtask of Dataflow A are 1, 2, and 1, respectively. Similarly, one ALU subtask in Dataflow B generates 2 product terms. Two product terms can be merged into one same ofmap pixel, while ifmaps or weights are not shareable. As a result, LDR of weights, ifmaps, and ofmaps in each ALU subtask of Dataflow B are 1, 1, and 2, respectively.

A register subtask is a combination of two ALU subtasks for both dataflows. In Dataflow A, despite LDR in one register subtask, an ofmap pixel stored in a register can be further shared by two different register subtasks, which means that ofmap LSR is twice larger than ofmap LDR while other LSRs are the same as LDRs. For example, without on-chip buffer accesses, the register storing p_1 in the first register subtask can be utilized again in the next register subtask. Similarly, the properties in a register subtask of Dataflow

B can be inferred once we notice that ifmap pixels can be shared by two different register subtasks in Dataflow B.

A buffer subtask corresponds to the total processing task for both dataflows. This means that a buffer subtask generates 8 product terms. An ifmap pixel in a buffer subtask is shared by 4 multiplications, while weights still cannot be shared.

In order to estimate the cost of data movement without hardware implementation, we evaluate the memory capacity and the memory access according to LSR and LDR in each subtask. Tab. 3.3 also introduces the memory capacity and the number of memory accesses in both dataflows, which will be explained next.

The register capacity can be estimated according to the number of operations in a register subtask and LDR. For example, in Dataflow A, the number of operations is 4 in a register subtask. Since weight LDR is 1 in a register subtask, there are 4 weights required to be stored in registers to process a register subtask. Similarly, other capacities can also be evaluated by LDR and the number of operations in a buffer subtask.

We are able to evaluate the number of register accesses according to LDR in an ALU subtask and the total number of operations. The total number of operations is 8. Once we notice that weight LDR of an ALU subtask in Dataflow A is 1, we can infer the total number of accesses to weight registers is 8 since each operation in an ALU subtask requires one access to the weight register. We further infer the total number of accesses to ifmap registers and ofmap registers through LDR in an ALU subtask. Similarly, the total number of accesses to on-chip buffers can be evaluated by LSR in registers and the total number of operations. According to the properties introduced in Tab. 3.3, we are able to evaluate the overall cost of data movement, thereby evaluating power, performance, and area.

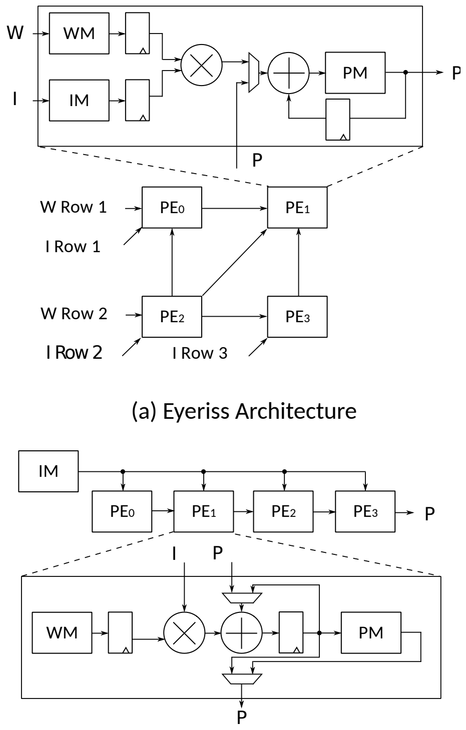
3.3.2 Related Work

Many previous studies have done a lot of work [22, 24–33] on off-chip access minimization focusing on CONV data reuse and partial sum (psum) accumulations. MEC [33] is a novel GPU-based processing method that utilizes data reuse. However, a GPU-based design is difficult to reduce the buffer capacity exploiting space reuse. Fig. 3.5 introduces two representative CNN acceleration ASICs. WM, IM, and PM in the figure refer to the on-chip buffers for weights, ifmaps, and psums, respectively. Eyeriss [27] in Fig. 3.5 (a) introduces a row-stationary (RS) processing dataflow. Each Processing Element (PE) has its own independent WM, IM, PM. Each PE computes psums between a weight row and an input row. The weight row is shared among PEs horizontally, the input row is shared diagonally and psums are accumulated vertically. In an ideal situation, RS dataflow can make full use of ifmap pixel and weight reuse to bring minimized off-chip memory access and high throughput. In order to adapt to different CONV shapes, PEs in RS are designed as computing units that work independently to maintain flexibility. As a result, each PE stores a small number of inputs locally for data reuse in the near future, even if adjacent PEs utilize the same set of inputs. The redundant buffer design leads to energy loss in on-chip buffers.

Considering the redundant ifmap and weight accesses, DSIP architecture [55] in Fig. 3.5 (b) calculates multiplications required by an ifmap pixel as quickly as possible. A single ifmap pixel broadcasts to all PEs, multiplied by all required weights and temporarily stored at registers in the PE array. The psum corresponding to the weight of the same row will be temporarily stored in PM in the PE. When the result of the next ifmap pixel arrives, it unicasts to the neighboring PE for accumulation. By using ifmap more efficiently, DSIP achieves better energy performance than Eyeriss. However, since the space reuse capability of psum in DSIP is limited to the same row, the space reuse capability among rows is ignored. As a result, it brings H_{w} times larger PM capacity than the case fully considered the space reuse. Furthermore, since weight size in a single CONV is usually small, in order to enhance throughput, DSIP expands its structure to multiple ofmap channels to further improve the utilization of ifmaps [55]. However, this kind of parallel computation among ofmap channels cannot effectively use the space reuse capability among ifmap channels we discussed above. In the worst case, DSIP will request an extra $H_{\mathrm{w}} \times C_{\mathrm{o}} \times$ larger PM capacity than the case that considers the space reuse, which leads to an energy overhead.

Both previous approaches aim to reduce the number of memory accesses by exploiting the data reuse, while the effect of the dataflow on on-chip memory capacity is rarely discussed. Since the capacity of the memory itself has a significant impact on the energy of single memory access, we can further improve energy efficiency by exploiting the space reuse in CONV to reduce memory capacity.

Several evaluation models are available to describe hardware performance. Accelergy [5] proposes a hardware configuration language to describe the processing architecture and relies on automated design exploration tools to calculate the energy. Interstellar [6] also uses a domain-specific language and presents an evaluation framework. Timeloop [7], Maestro [8] and Interstellar [6] utilize a loop-based method to systematically analyze the performance of DNN accelerators. DNN-chip predictor [9] proposes a fast analytical model under the fixed hardware settings such as the fixed memory capacity. However, previous works is less efficient to estimate required memory capacities according to the given dataflow. For example, DNN-chip predictor needs to traverse all possible scenarios



(a) Eyeriss Architecture

(b) DSIP Architecture

Figure 3.5: Previous Works: (a) Eyeriss Architecture [27] computes psums with the same row of weights in each PE, and accumulates psums by passing results to upper nearby PEs. (b) DSIP architecture[55] broadcasts ifmap pixel to all PEs and accumulates psums by passing results to next PEs.

possible to find an efficient memory capacity combination for one dataflow. Once the memory hierarchy becomes complex, it will be difficult for previous works to quickly and efficiently predict the best candidate dataflow.

In the next section, this chapter utilizes proposed LDR and LSR to design an energy-efficient CNN accelerator by analyzing LDR and LSR in the hardware architecture of the state-of-the-art accelerator DSIP [55].

3.3.3 Proposed Dataflow

This section proposes the CNN accelerator that takes full advantage of LDR and LSR.

In order to take into account both the local data reuse and the local space reuse capabilities, the energy-efficient 2D CONV dataflow must take into account both the horizontal and vertical reuse of weight, and also the horizontal and vertical reuse of ifmaps. Fig. 3.6 (a) introduces an example of the ifmap access pattern of DSIP when the weight size is 2×2 with two ofmap channels. Previous CNN accelerators aim to exploit the ifmap data reuse in CONV. Hence, DSIP dataflow is to process operations between one single ifmap pixel and all weights among two different ofmap channels in parallel. All weights are combined into one weight group (WG) and one ifmap pixel is one ifmap group (IG). Once the data are loaded into registers, DSIP dataflow fetches all weights in one WG and 1 ifmap pixel in one IG to fulfill all 8 multipliers per clock cycle. At the first clock cycle, the ifmap pixel a is sent to all multipliers to exploit the data reuse of ifmaps. After the first clock cycle, b, c, and d will be processed one by one until all ifmap pixels are processed in the DSIP dataflow. As shown in Fig. 3.6, (IG0 / WG0), (IG1 / WG0), (IG2 / WG0) and (IG3 / WG0) are processed sequentially. One DSIP drawback is that the local space reuse is not available in the ALU subtask, since psums among different ofmap channels can not be accumulated. This forces the DSIP accelerator must reserve enough on-chip storage space to store the intermediate psums avoiding expensive off-chip memory accesses.

To solve this issue, the proposed dataflow tries to exploit both the local data reuse and the local space reuse at the same time within the ALU subtask. The proposed dataflow accesses ifmap pixels concurrently from multi rows in Fig. 3.6 (b). The weights in one ofmap channel are combined into one WG and two ifmap pixels are combined into one IG. At the first clock cycle, ifmap pixels a and e are multicasted to weights in 2D CONV to exploit the data reuse in multiplications once accessed. At the next clock, b and f are processed. This ensures that several product items can be accumulated into psums in a single clock cycle. Compared to DSIP dataflow, the proposed dataflow reduces the number of psum register accesses and the required psum register capacity. When the accumulation within 2D CONV is finished, the proposed dataflow processes 2D CONVs one ifmap channel by one ifmap channel, thereby reducing the required memory space for psums. The processing among ofmap channels is left at the end since there is no space reuse available. By considering both local space reuse and local data reuse, the proposed dataflow is able to reduce the number of memory accesses while reducing the required memory capacity.

More specifically, **Algorithm** 1 introduces the convolution processing dataflow based on DSIP [55]. The fifth row introduces the range of an ALU subtask. The fourth and fifth rows introduce the range of a register subtask. The third, fourth, and fifth rows introduce the range of a buffer subtask. The second, third, fourth, and fifth rows introduce the range of a memory subtask. DSIP ALU subtask is to process operations between one ifmap pixel and all $H_\mathrm{w}W_\mathrm{w}$ weights inside 2D CONV in parallel. In actual convolution processing, since the number of weights varies, it is difficult to make full use of all ALUs if weights in only one channel are loaded. Therefore, the ALU subtask processes weights from multiple β ofmap channels in one single cycle. The parameter β is scaled from 1 to C_o in order

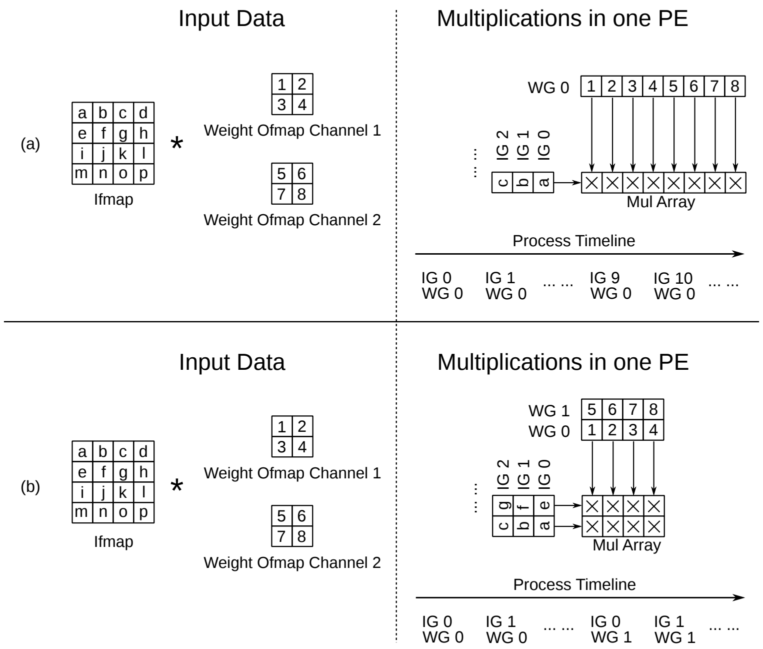


Figure 3.6: The ifmap data reuse capability and the ofmap space reuse capability in (a) DSIP and (b) the proposed dataflow. DSIP fetches one ifmap pixel and processes the pixel among ofmap channels. Although the ifmap data reuse is maximized, psums with no space reuse capability are generated. The proposed work fetches ifmap pixels in the same column, generates psums which can be added immediately thus saving memory capacity.

to improve ALU utilization. The parameter β also affects the number of operations in a buffer subtask. In order to make full use of on-chip buffers, the processing dataflow scales γ from 1 to W_{o} to adjust the capacity of the on-chip buffer, which is shown in the fourth row. The number of ALUs is $H_{\mathrm{w}}W_{\mathrm{w}}\beta$. According to data loading from registers to ALU arrays, LDR for ifmaps is enhanced to $H_{\mathrm{w}}W_{\mathrm{w}}\beta$ in a register subtask. Weight LDR is γ in a register subtask. At the same time, the product terms in a register subtask can be accumulated for W_{w} ×. DSIP dataflow exploits LDR of ifmaps in an ALU subtask. In DSIP, however, LDR of ofmaps and weights are not well utilized. This forces the accelerator to reserve enough on-chip buffer for product terms and weights for avoiding expensive off-chip memory accesses.

The proposed dataflow aims to reduce the on-chip memory capacity for ofmaps and weights. **Algorithm** 2 explains the details of the proposed convolution dataflow. The fifth row introduces the range of an ALU subtask. The fourth and fifth rows introduce the range of a register subtask. The third, fourth, and fifth rows introduce the range of a buffer subtask. The second, third, fourth, and fifth rows introduce the range of a memory subtask. The dataflow accesses ifmap pixels concurrently on γ columns and δ rows as shown in the fifth row. Ifmap pixels are multicasted to all weights in one channel to ensure LDR of

Algorithm 1 Processing dataflow based on DSIP [55]

```
Require: $W, I$
Ensure: $O$
Initialization;
for $x_1 \in [1 : W_o/\gamma]$ do
    for $k_{\in}[1 : C_i], u_1 \in [1 : C_o/\beta], y \in [1 : H_o]$ do
        for $x_2 \in [1 : \gamma]$ do
            for $u_2 \in [1 : \beta], i \in [1 : W_w], j \in [1 : H_w]$ in parallel do
                $\mathbf{O}[u_1u_2][x_1x_2][y] += \mathbf{I}[k][x_1x_2 + i][y + j] \times \mathbf{W}[u_1u_2][k][i][j]$;
            end for
        end for
    end for
end for
Return $O$
```

Algorithm 2 Processing dataflow for convolution.

```
Require: $W, I$
Ensure: $O$
Initialization;
for $k_1 \in [1 : C_i/\alpha], u_1 \in [1 : C_o/\beta]$ do
    for $k_2 \in [1 : \alpha], u_2 \in [1 : \beta], y_1 \in [1 : H_o/\delta]$ do
        for $x_1 \in [1 : W_o/\gamma]$ do
            for $x_2 \in [1 : \gamma], y_2 \in [1 : \delta], i \in [1 : W_w], j \in [1 : H_w]$ in parallel do
                $\mathbf{O}[u_1u_2][x_1x_2][y_1y_2] += \mathbf{I}[k_1k_2][x_1x_2 + i][y_1y_2 + j] \times \mathbf{W}[u_1u_2][k_1k_2][i][j]$;
            end for
        end for
    end for
end for
Return $O$
```

ifmaps in ALUs. Similarly, in order to maintain ALU utilization, the dataflow is able to process ifmap pixels from δ rows and γ columns in an ALU subtask. In a register subtask, the dataflow finishes all operations to increase LDR for ifmaps, weights, and ofmaps. LDR of ofmaps is increased within a buffer subtask. As a result, all product terms are merged into final ofmap pixels without sending partial sums to the off-chip memory. Furthermore, the dataflow stores data among α ifmap channels and β ofmap channels on-chip to adopt the size of the on-chip buffer.

In ALU subtasks and register subtasks, the processing proposed dataflow improves LDR for ifmaps, weights, and ofmaps. Although the on-chip buffer in the proposal has better LDR for weights and ofmaps, it suffers a large ifmap buffer capacity overhead. The proposed dataflow could perform well if the cost of data movement for ifmap pixels does

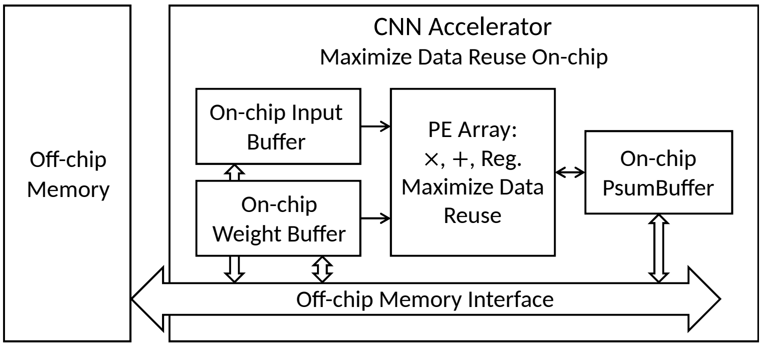

CNN Accelerator
Maximize Data Reuse On-chip
Off-chip Memory
On-chip Input Buffer
On-chip Weight Buffer
PE Array: ×, +, Reg. Maximize Data Reuse
On-chip PsumBuffer
Off-chip Memory Interface

Figure 3.7: Accelerator architecture: Parallel MACs are processed in the proposed CNN accelerator. On-chip buffers for weight and ifmap store the data, waiting for data reuse in multiplications. The PE array contains multiple multipliers, adders and registers to accumulate psums as much as possible. Calculated ofmap pixels are sent directly to the off-chip memory through the off-chip memory interface. The PM stores psums which cannot be accumulated into the ofmap pixels in time.

not come to be dominant.

The proposed architecture shown in Fig. 3.7 aims to reduce the required on-chip buffer capacity while also limiting the number of accesses to the on-chip buffer according to data reuse. In order to implement the proposed dataflow in 2D CONV, a potential choice is to use $H_\mathrm{w} \times W_\mathrm{w} \times H_\mathrm{o}$ PEs for parallel calculations thus no more PM is required in order to maximize the utilization of ofmap LDR and LSR. By psum accumulations for W_o clock cycles, ofmap LDR and LSR are exploited at the same time. However, the ifmap height H_o is sometimes too large to be implemented in the hardware design and sometimes too small to fully utilize available multipliers in the PE array. As a compromise, the proposed architecture chooses a value n which can be scaled from 1 to H_o to exploit the LSR in 2D CONV. Therefore, the PM is utilized to store psums among ifmap channels which cannot be quickly accumulated into ofmap pixels due to PE scale issues.

Fig. 3.8 introduces the basic proposed PE array with n set to 1. When n is 1, all psums in one clock cycle correspond to different ofmap spaces. At this time, psums which require accumulations are temporarily stored in the register array. When the next ifmap pixel is calculated, adders accumulate the current psums into the psum registers. If accumulation results require the pixel of next ifmap rows, the PE array sends the results to PM. When the access of ifmap comes to the next row, the previous psums stored in PM is re-read to be accumulated into final ofmap pixels.

Fig. 3.9 introduces the proposed PE array with n set to 2. When n is equal to 2, we can utilize the reuse capability under one clock cycle, add related psum and send it to psum register. Psums which are calculated into final results are sent directly to off-chip memory.

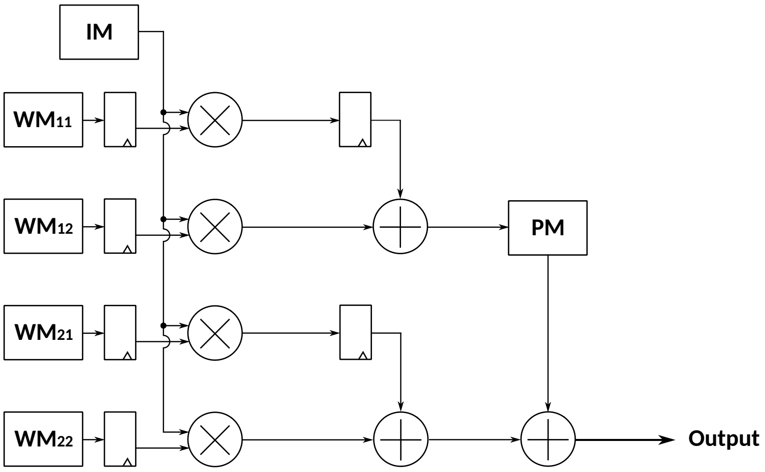


Figure 3.8: PE array architecture: Reduce the **capacity** of the PM and psum registers by adding psums as soon as possible. The weight size is assumed to be 2×2. IM, OM capacity is equal to the input width. All weights are stored in WM.

Table 3.4: Parameters for Comparison.

a	Filter Height / Width
b	Ifmap Height / Width
N	Ofmap Channel
Na^2	PE amount

This reduces both the number of accesses to PM and the required capacity of PM.

Through the simultaneous reuse of LDR and LSR, the proposed dataflow achieves the same or even higher throughput with less memory capacity and memory accesses. In the next section, this chapter compares and analyzes quantitatively the similarities and differences between the proposed architecture and DSIP architecture under 2D CONV conditions.

3.4 Experimental Results for the Proposed Dataflow

In this section, we compare the hardware designs based on the proposed dataflow with the DSIP dataflow to verify the efficiency of the evaluation based on LDR and LSR. Since both ALU subtasks are restricted in one ifmap channel, we set the comparison environment to the convolution operation with a single-ifmap channel, and multi-ofmap channels for simplicity. The detailed convolution shapes are listed in Tab. 3.4.

Tab. 3.5 introduces the register requirements for the convolution with N ofmap channels with an ifmap size of b^2 with a weight size of a^2. The memory requirements for DSIP

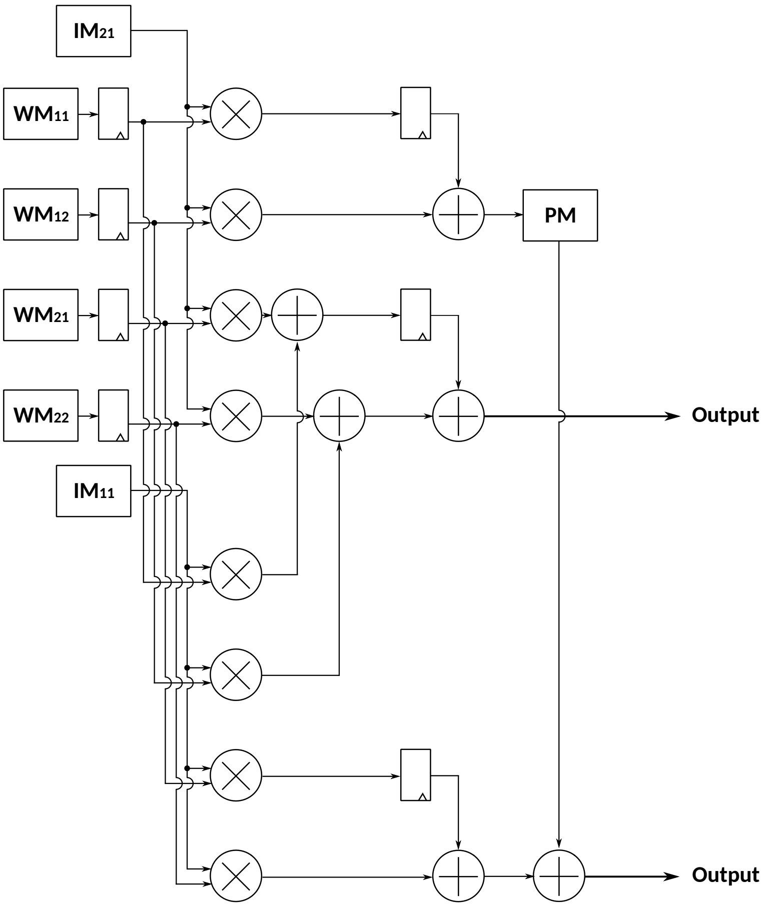


Figure 3.9: PE array architecture with n equal to 2. The weight size is assumed to be 2×2. Most psums are able to be accumulated into final ofmap pixels in register arrays. This reduces both PM access and PM capacity requirement.

in the $N = 1$ case have been summarized in Ref. [25], which describes the details for the processing dataflow in DSIP. We extend it to the case of $N > 1$ here. DSIP is committed to calculating all relevant multiplications for one pixel of an ifmap once. In the case of N ofmap channels, DSIP requires Na^2 PEs. The required sizes of weight registers and psum registers are proportional to the number of PE. Therefore, Na^2 weight registers and psum registers are required. An ifmap register directly broadcasts the pixel to all PEs, thus no register is required. Considering the number of register accesses, DSIP and proposed architecture both use weight stationary architecture, thus the number of visits to the weight register is equivalent to the order of processing time b^2. In the evaluation of this chapter, we set $n = N$ to guarantee that the number of ALUs in DSIP is the same as the one in the proposal. As a result, the proposed one utilizes the vertical reuse capability in 2D CONV to merge a vertical psums into one for psum registers. This is consistent with the conclusion in Section 3.3.3 that the proposed architecture requires less psum registers.

Table 3.5: Minimum register requirement for the acceleration dataflow. Order of processing time for both architecture is b^2.

Architecture	PE	Order of Reg. Size		Order of Reg. Access	
		Filter	Psum	Filter	Psum
DSIP [55]	Na^2	Na^2	Na^2	b^2	Na^2b^2
This work	Na^2	a^2	$a^2 + aN$	b^2	Nab^2

Table 3.6: Minimum buffer requirement for the acceleration dataflow.

Architecture	Order of Buf. Size			Order of Buf. Access		
	Ifmap	Filter	Psum	Ifmap	Filter	Psum
DSIP [55]	b	Na^2	Na^2b	Nb^2	Na^2	Nab^2
This work	Nb	a^2	a^2b	Nb^2	Na^2	ab^2

Tab. 3.6 introduces the on-chip buffer requirements. DSIP calculates in parallel among ofmap channels, while this work calculates in parallel inside a single ofmap channel. For ifmap pixels, this work requires an extra $N \times$ larger IM to perform parallel calculations due to ifmap LDR issue. However, the parallel computation among ofmap channel does not have any possible data reuse, so DSIP needs $N \times$ larger WM and PM. Considering the number of × the buffer is accessed, since LDR that DSIP can utilize is the horizontal accumulation of weight, it needs to access the PM Nab^2 times.

To prove the superiority of the proposed structure, we compare the energy consumption of the proposed work with the state-of-the-art accelerator DSIP [55] in Register-Transfer Level (RTL) simulation. We collect energy consumption for multiplication-accumulate calculations (MACs) and registers from a standard cell library in 65-nm process technologies. The energy consumption of on-chip buffer and off-chip memory access is collected from the SRAM model in CACTI Ver. 7.0 [15] under 65nm process with 1.2V. Tab. 3.7 summarizes energy measured in 1.2V supply voltage. All candidates are built in 16-bit fixed-point representation. Code tracing logs are generated to record activities through the RTL simulation for both accelerators running benchmarks. We simulate the energy consumption through the activities introduced in Eq. (3.3).

$$\begin{aligned} E =& E_{\mathrm{MAC}} * C_{\mathrm{MAC}} + E_{\mathrm{REG}} * C_{\mathrm{REG}} \\ &+ E_{\mathrm{BUF}} * C_{\mathrm{BUF}} + E_{\mathrm{OFF}} * C_{\mathrm{OFF}} \end{aligned} \tag{3.3}$$

where E, E_{MAC}, E_{REG}, E_{BUF}, and E_{OFF} are the total energy, the energy of single MAC process, single access to registers, single access to on-chip buffers and single access to off-chip memories, respectively. C_{MAC}, C_{REG}, C_{BUF}, and C_{OFF} are the total access count

Table 3.7: Related Energy Consumption in Convolution Operations.

Component		Energy
16-bit Register Access* (E_{REG})		0.18 pJ
16-bit Multiply Operation (E_{MAC})		0.21 pJ
16-bit On-chip Buffer Access* (E_{BUF})	1 kB	2.04 pJ
	8 kB	6.63 pJ
16-bit 8MB Off-chip SRAM Access (E_{OFF})		104.45 pJ

*: Average energy consumption (read and write operations).

of MAC processes, registers, on-chip buffers and off-chip memories, respectively.

Based on Eq. (3.3) and Tables 3.5, 3.6 and 3.7, we simulate the energy consumption of DSIP and the work in this chapter. The number of ofmap channels is set to 30.

Based on the RTL simulation for both dataflows, Fig. 3.10 introduces the buffer capacity requirement to minimize the off-chip memory access when processing convolution layers in Alexnet. The proposed dataflow requires to store data for all processing tasks in all ifmap channels and at least one ofmap channel. Therefore, there is a capacity overhead to store ifmaps in the on-chip buffer. Similarly, DSIP dataflow requires to store data in all ofmap channels and at least one ifmap channel, which leads to a capacity overhead of ofmaps. The minimum buffer requirement of the proposed work is 2.56× smaller than the buffer requirement for DSIP, since the number of ofmap channels is larger than the number of ifmap channels in Alexnet CONV layers.

Fig. 3.11 introduces the energy consumption of DSIP and the proposed work with 64 multipliers when processing convolution layers in Alexnet. The proposed work achieves 2.71× energy reduction by limiting the memory space to store psums. At the same time, the proposed architecture guarantees that most psums can be accumulated into final ofmap pixels without access to on-chip memories. By considering LDR and LSR, the proposed architecture considers both the required buffer capacity and the number of buffer accesses. Furthermore, the proposed architecture reduces both the required buffer capacity and the required energy consumption.

Fig. 3.12 introduces the energy consumption of DSIP and the proposed work with various PE amounts. The proposed work achieves 1.64× energy reduction by removing the unnecessary PM space. When N is more than 1, the energy consumption of on-chip memories comes to be at least $N\times$ smaller than DSIP. This architecture guarantees that most psums can be accumulated into the final ofmap pixel without access to the PM. Therefore, this work dramatically reduced the access of the PM to ab^2 times. Energy consumption of off-chip memory does not change since both architectures reduce off-chip memory accesses.

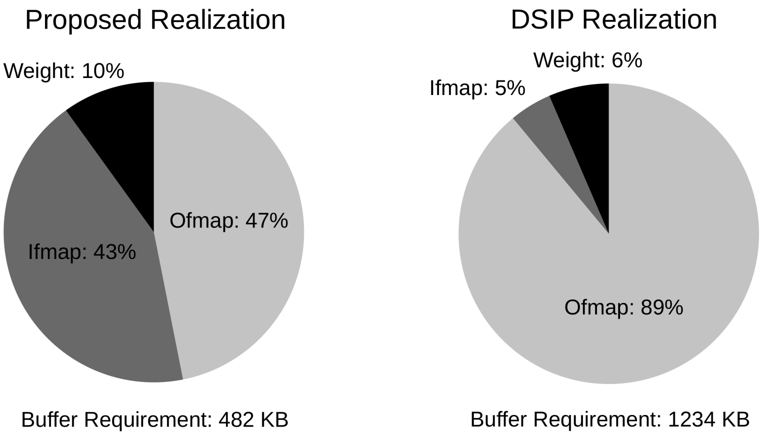


Figure 3.10: On-chip buffer requirements to minimize off-chip memory access for CONV processing tasks in Alexnet.

Fig. 3.13 introduces the energy consumption of the proposed architecture and DSIP with various ifmap sizes. Since the change of ifmap size b^2 does not change the proportional relationship of LDR or LSR in the convolution operation, the ifmap size does not affect the proportional relationship of the energy consumption between two structures. Fig. 3.14 introduces the energy consumption of the proposed architecture and DSIP with various weight sizes. It proves that the rates of reuse are directly proportional to the weight size.

In order to further prove the scalability and energy consumption superiority of the structure in this chapter in a practical environment, we design the proposed accelerator using the hardware description language with different numbers of multipliers. We still choose Alexnet [56] as the CNN evaluation model because of its variable weights, ifmap size and channels. In the experiment, a batch of 4 images is processed. The performance of the proposed architecture is introduced in Tab. 3.8. Although DSIP theoretically optimizes read access to the ifmap pixel, it ignores the potential LDR and LSR in the ofmap pixel. For 2D CONV under a single channel, we theoretically need only 4.2 kB on-chip buffers to maximize the reuse. For resource reuse among channels, however, we set the capacity of the on-chip buffer to 60.4 kB for reducing the number of off-chip accesses to the same level of DSIP one. We achieve 2.30× less on-chip buffer and 2.18× energy reduction while obtaining similar throughput. Since the proposed structure can exploit the reuse among ifmap channels, this architecture can easily be extended to a large PE scale which is no larger than $H_{\mathrm{o}} \times W_{\mathrm{o}} \times H_{\mathrm{w}} \times W_{\mathrm{w}} \times C_{\mathrm{i}}$. Following the same dataflow, we also designed a larger-scale architecture and compares the performance with Eyeriss v2 [27]. The results show that we can get 7.45× on-chip buffer reduction and 4.67× energy reduction against

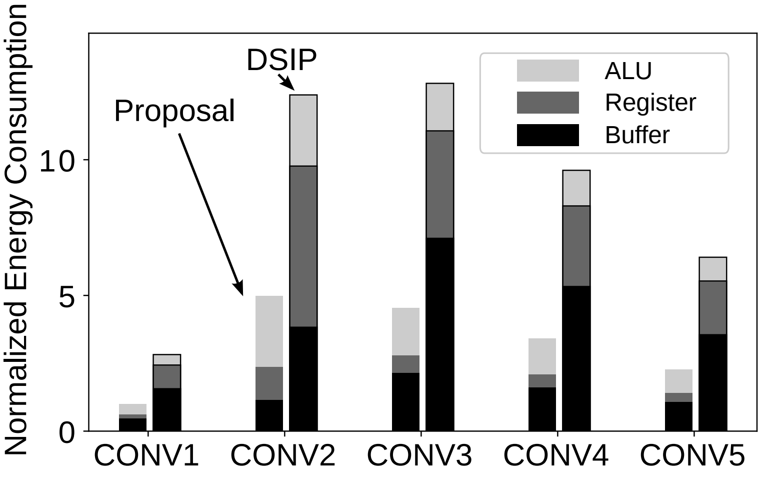


Figure 3.11: Verification for the proposed metrics with CONV processing tasks in Alexnet. All X axises consist of five CONV layers in Alexnet. Y axises refer to the normalized energy consumption. Energy consumption is normalized by the simulation result of the proposed work in CONV1.

Table 3.8: Accelerator Performances for Alexnet.

Architecture	DSIP [55]	This	Eyeriss2 [27]	This
Number of PEs	64	72	384	432
Frequency [MHZ]	250	250	200	200
Throughput [GMACS]	11.7	13.9	61.8	73.8
On-chip Memory [kB]	139.6	60.7	492*	66.0
Power [mW]	93.4	42.9	450	96.4

On-chip memory consists of all buffers and registers.
*: Memory capacity is normalized in 16 bit.

Eyeriss v2.

Based on the analysis of LDR and LSR in subtasks, this chapter designed a dataflow that is more energy-efficient than the state-of-the-art CNN processing dataflow [55]. with the analysis based on the hardware implementations, the evaluation results of accelerators are consistent with those based on LDR and LSR. Therefore, the evaluation considering both LDR and LSR can help us design more energy-efficient DNN accelerators without detailed hardware implementations.

Furthermore, it is necessary to establish an analytical model that can predict a better dataflow and the memory capacity requirement at the same time. Based on the proposed hardware properties, this chapter first builds an analytical evaluation model that can estimate the dataflow performance and the corresponding memory capacity requirement for each dataflow. In the next section, we utilize *local data reuse* and *local space reuse* to establish hardware metrics of power, performance, and area.

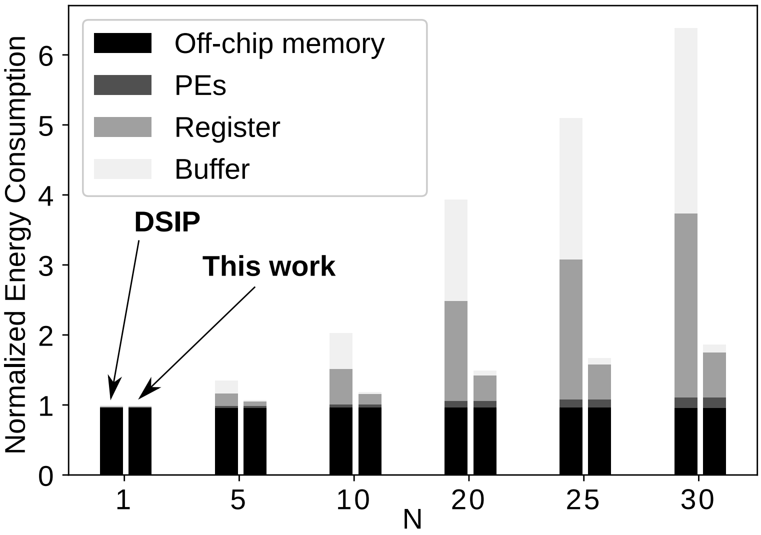


Figure 3.12: Energy comparison between DSIP and proposal with PE amount Na^2. Energy is normalized by DSIP energy consumption with $N == 1$. The ifmap size b^2 is set to 100^2. The weight size a^2 is set to 5^2.

3.5 Analytical Hardware Evaluation Model

We further propose analytical metrics for energy, time, and area of a corresponding hardware implementation according to *local data reuse* and *local space reuse* of subtasks in a given processing dataflow.

In an actual hardware implementation, the cost of memory with respect to area and energy differs depending on the type of the memory, the capacity of the memory, as well as its manufacturing process. We introduce parameters k_{Energy}, k_{Time}, and k_{Area}, which represent an energy of one memory access, an access time to a memory, and the area of one memory macro, respectively. Furthermore, we use parameter y for identifying a particular subtask in the memory hierarchy consisting of “registers”, “on-chip buffers”, and “off-chip memories”. We use parameter x to represent a category of stored data which includes “weight”, “ifmap”, and “ofmap”.

Analytical Evaluation for Energy

In DNN accelerators, data movement dominates the energy consumption of an entire architecture. Therefore, we evaluate energy metric focusing only on memories and ALUs instead of an entire hardware. The energy consumption is thus expressed as a sum of energy consumption in memories and ALUs including each memory for different data

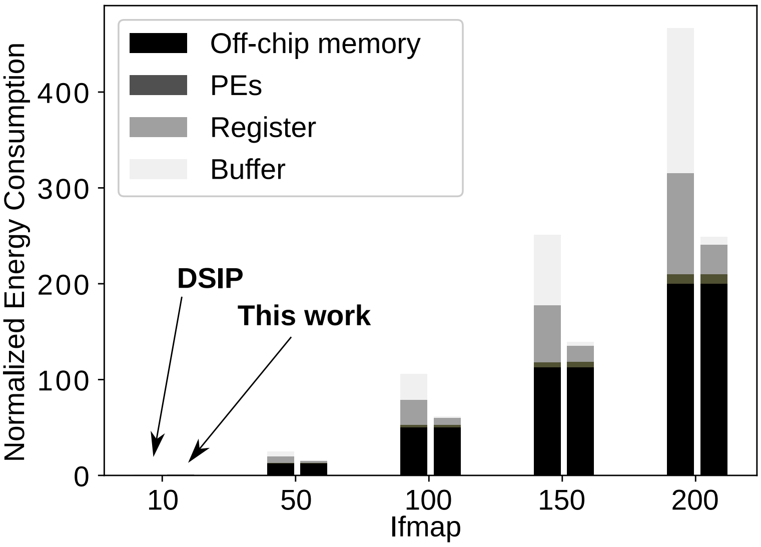


Figure 3.13: Energy comparison between DSIP and proposal with variable I, normalized by DSIP energy consumption with the ifmap size b^2 set to 10^2. N is set to 10. The weight size a^2 is set to 5^2.

types x and different subtasks y, which is expressed as:

$$E = E^{\mathrm{ALU}} + \sum_{x \in \{\mathrm{W,I,O}\}} \sum_{y \in \{\mathrm{Reg,Buf,Off}\}} E_x^y, \tag{3.4}$$

where E, E^{ALU} and E_x^y are the energy metric of the entire hardware, the energy consumption of ALUs, and the consumed energy of a memory for data type x in memory level y, respectively.

For an ALU array, the energy consumption can be derived from the total number of operations N^{all} in the whole DNN processing and the energy per operation. The energy per operation is defined as $k_{\mathrm{Energy}}^{\mathrm{ALU}}$. Therefore, the energy consumption of ALUs is expressed as:

$$E^{\mathrm{ALU}} = k_{\mathrm{Energy}}^{\mathrm{ALU}} N^{\mathrm{all}}. \tag{3.5}$$

For memories, the energy consumption is derived from the number of accesses and the energy per access. In order to facilitate formulas to represent the number of memory accesses, we define levels of processing at ALUs, registers, on-chip buffers, and off-chip memory as Level 0 (L0), Level 1 (L1), and Level 2 (L2), and Level 3 (L3), respectively. In the following, we use parameters D_x^y and S_x^y which represent LDR and LSR of data type x at subtask y, respectively. With those parameters, we can evaluate the number of L3 accesses by dividing the total number of operations N^{all} by L2 LSR S_x^2. Similarly, the

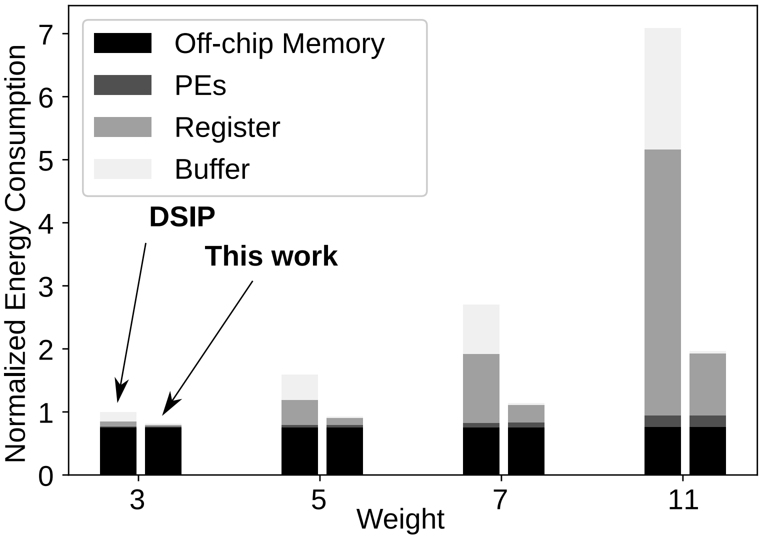


Figure 3.14: Energy comparison between DSIP and proposal with variable W, normalized by DSIP energy consumption with weight size 3^2. The ifmap size b^2 is set to 100^2. N is set to 10.

number of L2 accesses can be calculated by similar methods. For the register case, we evaluate the number of L1 accesses by dividing the total number of operations N^{all} by L0 LDR D_x^0, since L0 ALUs have no memory to reuse data. For simplicity of notation, we define that ALU LSR is the same as ALU LDR. Therefore, the number of accesses to every memory level y can be evaluated by dividing the total number of operations N^{all} by LSR S_x^{y-1}.

The energy per access to memories can be designed with different hardware structures. For example, the energy per access to an on-chip memory is determined by its required capacity while the energy per access to an off-chip memory is constant regardless of the dataflow implemented in the hardware. The energy per access to an on-chip memory $E_{x_{\text{access}}}^y$ can be roughly estimated by the square root of its capacity $\sqrt{C_x^y}$ [15], where C_x^y represents the capacity of memory x in memory level y. Therefore, the energy consumption of on-chip memory x in level y can be expressed as:

$$E_x^y = \frac{N^{\text{all}}}{S_x^{y-1}} \times E_{x_{\text{access}}}^y, \qquad E_{x_{\text{access}}}^y = k_{\text{Energy}}^y \sqrt{C_x^y}, \tag{3.6}$$

where E_x^y, and $E_{x_{\text{access}}}^y$ are the energy consumption of a memory for data type x in memory level y, and the energy consumption per access to a memory for data type x in memory level y, respectively.

The energy consumption of off-chip memory x can be expressed as:

$$E_x^{\mathrm{Off}} = \frac{N^{\mathrm{all}}}{S_x^{\mathrm{Buf}}} \times k_{\mathrm{Energy}}^{\mathrm{Off}}, \tag{3.7}$$

where E_x^{Off}, and $k_{\mathrm{Energy}}^{\mathrm{Off}}$ are the energy consumption of an off-chip memory for data type x, and the energy parameter for one access to an off-chip memory, respectively.

The memory capacity is affected by the number of operations and LDR in each subtask. We introduce parameter N^y which represents the number of operations in a subtask at level y. The required number of data can be inferred from the number of operations in a subtask N^y divided by LDR D_{x}^y in each subtask. Therefore, the memory capacity can be expressed as:

$$C_x^y = \frac{N^y}{D_x^y}. \tag{3.8}$$

Analytical Evaluation for Time

In DNN processing, the throughput is affected by the processing time consumed in an ALU array and the data transfer time among memories. The processing time of an ALU array is inferred by the total number of operations N^{all} and the number of operations in an ALU subtask N^{ALU} that can be processed in one clock cycle. Therefore, the total transfer time is the sum of the transfer time between off-chip memories and on-chip buffers and the time between on-chip buffers and registers.

We use parameter $k_{\mathrm{Time}}^{\mathrm{ALU}}$ that represents one clock cycle including data accessing time and either of multiply and multiply-add operations. The number of data that can be transferred in parallel between on-chip buffers and registers is determined by the bandwidth of data transfer network between on-chip buffers and registers, which is represented by the parameter B^{Buf}. We assume that multiple clock cycles are required for on-chip buffer access and the time required for on-chip buffer access is represented by the parameter $k_{\mathrm{Time}}^{\mathrm{Buf}}$. In the data transfer of a register subtask, even if the number of data to be transferred is less than the bandwidth, one access time for buffer access is still required. Therefore, we use the ceiling function $\lceil\ \rceil$ to represent the transfer time of register subtasks. Similarly, the bandwidth of off-chip memories is represented by the parameter B^{Off}. Combining processing time in ALUs and transfer time in memories, we refer the time metric as:

$$T = k_{\mathrm{Time}}^{\mathrm{ALU}} \frac{N^{\mathrm{all}}}{N^{\mathrm{ALU}}} + \sum_{x \in \{\mathrm{W,I,O}\}} \sum_{y \in \{\mathrm{Buf,Off}\}} \frac{k_{\mathrm{Time}}^{y} N^{\mathrm{all}}}{N^{y-1}} \left\lceil \frac{N^{y-1}}{B^y S_x^{y-1}} \right\rceil, \tag{3.9}$$

where T is the metric of the time consumption.

Analytical Evaluation for Area

The capacity of a memory roughly determines the area of a memory. We utilize memory capacity in Eq.(3.10) and the area parameter of memories k^{y}_{Area} to estimate a memory area. Besides the area consumed by memories, the area consumption of ALUs should also be considered. We utilize the number of operations in ALU subtask N^{ALU} and the area parameter of ALUs $k^{\text{ALU}}_{\text{Area}}$ to represent the area of ALU arrays. Therefore, for the processing hardware, its on-chip area metric can be expressed as:

$$\begin{aligned} A &= A^{\text{ALU}} + \sum_{x\in\{\text{W,I,O}\}} \sum_{y\in\{\text{Reg,Buf}\}} A^{y}_{x}, \\ A^{\text{ALU}} &= k^{\text{ALU}}_{\text{Area}} N^{\text{ALU}}, \quad A^{y}_{x} = k^{y}_{\text{Area}} C^{y}_{x}, \end{aligned} \tag{3.10}$$

where A, A_{ALU}, and A^{y}_{x}, are the total area of the whole processing architecture, the area of the ALU array, and the area of data type x in memory level y, respectively.

In the following sections, we utilize the proposed metrics to evaluate the proposed accelerator and a state-of-the-art accelerator [55] for various processing tasks to verify the accuracy and the flexibility of the proposed hardware metrics.

3.6 Dataflow Comparison

According to LDR and LSR in each subtask, we are able to compare time, energy, and area of a hardware implementation. In this section, we investigate subtasks in different processing dataflows and describe LDR and LSR in subtasks of the processing dataflows based on the proposed accelerator and state-of-the-art accelerators [24, 55]. The processing for biases is omitted here for simplicity.

3.6.1 Dataflow for Convolutions

We take the proposed accelerator and the DSIP accelerator [55] that handle convolution operations as examples. Evaluation metrics based on LDR and LSR make it possible to investigate different Performance-Power-Area trade-off characteristics of the two dataflows.

The detailed algorithms have been discussed in Section 3.3.3. Tabs. 3.9 and 3.10 introduce LDR and LSR in both dataflows when processing different convolutions. In ALU subtasks and register subtasks, the processing proposed dataflow improves LDR for

Table 3.9: Hardware properties in the proposed processing dataflow for convolution.

Subtask	N^y	Type	D_x^y	S_x^y
ALU	$H_\mathrm{w}W_\mathrm{w}\gamma\delta$	**W**	$\gamma\delta$	$\gamma\delta$
		I	$H_\mathrm{w}W_\mathrm{w}$	$H_\mathrm{w}W_\mathrm{w}$
		O	$H_\mathrm{w}W_\mathrm{w}$	$H_\mathrm{w}W_\mathrm{w}$
Reg	$H_\mathrm{w}W_\mathrm{w}W_\mathrm{o}\delta$	**W**	$W_\mathrm{o}\delta$	$H_\mathrm{o}W_\mathrm{o}$
		I	$H_\mathrm{w}W_\mathrm{w}$	$H_\mathrm{w}W_\mathrm{w}$
		O	$H_\mathrm{w}W_\mathrm{w}$	$H_\mathrm{w}W_\mathrm{w}$
Buf	$H_\mathrm{w}W_\mathrm{w}H_\mathrm{o}W_\mathrm{o}\alpha\beta$	**W**	$H_\mathrm{o}W_\mathrm{o}$	$H_\mathrm{o}W_\mathrm{o}$
		I	$H_\mathrm{w}W_\mathrm{w}\beta$	$H_\mathrm{w}W_\mathrm{w}\beta$
		O	$H_\mathrm{w}W_\mathrm{w}\alpha$	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$
Off	$C_\mathrm{i}C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}H_\mathrm{o}W_\mathrm{o}$	**W**	$H_\mathrm{o}W_\mathrm{o}$	$H_\mathrm{o}W_\mathrm{o}$
		I	$C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}$	$C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}$
		O	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$

Table 3.10: Hardware properties in the processing dataflow based on DSIP.

Subtask	N^y	Type	D_x^y	S_x^y
ALU	$H_\mathrm{w}W_\mathrm{w}\beta$	**W**	1	1
		I	$H_\mathrm{w}W_\mathrm{w}\beta$	$H_\mathrm{w}W_\mathrm{w}\beta$
		O	1	1
Reg	$H_\mathrm{w}W_\mathrm{w}\beta\gamma$	**W**	γ	$H_\mathrm{o}\gamma$
		I	$H_\mathrm{w}W_\mathrm{w}\beta$	$H_\mathrm{w}W_\mathrm{w}\beta$
		O	W_w	$H_\mathrm{w}W_\mathrm{w}$
Buf	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}H_\mathrm{o}\beta\gamma$	**W**	$H_\mathrm{o}\gamma$	$H_\mathrm{o}W_\mathrm{o}$
		I	$H_\mathrm{w}W_\mathrm{w}\beta$	$H_\mathrm{w}W_\mathrm{w}\beta$
		O	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$
Off	$C_\mathrm{i}C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}H_\mathrm{o}W_\mathrm{o}$	**W**	$H_\mathrm{o}W_\mathrm{o}$	$H_\mathrm{o}W_\mathrm{o}$
		I	$C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}$	$C_\mathrm{o}H_\mathrm{w}W_\mathrm{w}$
		O	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$	$C_\mathrm{i}H_\mathrm{w}W_\mathrm{w}$

ifmaps, weights, and ofmaps, while DSIP LDR for ifmaps is better than the proposal LDR. Considering the on-chip buffer, although the on-chip buffer in the proposal has better LDR for weights and ofmaps, it suffers a larger ifmap buffer capacity than DSIP. According to evaluation metrics, the DSIP processing dataflow is better when the cost of data movement for ifmap pixels comes to be dominant, while the proposed dataflow could be better in other cases.

If the bandwidth between on-chip buffers and registers is limited, the transfer time of ifmap pixels is dominant in the proposed dataflow while the transfer time of weights is dominant in the DSIP dataflow. If the bandwidth between off-chip memories and on-chip buffers is limited, similar discussion can be held.

Algorithm 3 Processing dataflow based on TPU [24]

```
Require: $W, I$
Ensure: $O$
1: Initialization;
2: for $k_1 \in [1 : C_\text{i}/\alpha]$ do
3:     for $u_1 \in [1 : C_\text{o}/\beta]$, $y_1 \in [1 : H_\text{o}/\delta]$ do
4:         for $k_2 \in [1 : \alpha]$, $u_2 \in [1 : \beta]$, $y_2 \in [1 : \gamma]$ in parallel do
5:             $\mathbf{O}[u_1u_2][y_1y_2] += \mathbf{I}[k_1k_2][y_1y_2] \times \mathbf{W}[u_1u_2][k_1k_2]$;
6:         end for
7:     end for
8: end for
9: Return $O$
```

Table 3.11: Hardware properties in the processing dataflow based on TPU.

Subtask	N^y	Type	D_x^y	S_x^y
ALU	$\alpha\beta\delta$	**W**	δ	δ
		I	β	β
		O	α	α
Buf	$C_\text{o}H_\text{o}\alpha$	**W**	H_o	H_o
		I	C_o	C_o
		O	α	C_i
Off	$C_\text{i}C_\text{o}H_\text{o}$	**W**	H_o	H_o
		I	C_o	C_o
		O	C_i	C_i

3.6.2 Dataflow for Matrix Multiplications

Tensor Processing Unit (TPU) is a general-purpose accelerator that has the ability to realize many kinds of processing dataflows [4, 28, 61]. **Algorithm** 3 introduces a processing dataflow which can be realized by TPU [24]. The fourth row introduces the range of an ALU subtask. The third and fourth rows introduce the range of a buffer subtask. The second, third, and fourth rows introduce the range of a memory subtask. An ALU subtask consists of multiple ifmap pixels from δ columns and α ifmap channels and multiple weights from α ifmap channels and β ofmap channels. Hardware properties of the TPU dataflow are summarized in Tab. 3.11. The number of ALUs is $\alpha\beta\delta$. According to the high scalability of an ALU subtask, the dataflow can change the processing range to meet the shape of the matrix.

Another example of a processing dataflow for the matrix multiplication is listed in **Algorithm** 4 which is taken from the proposed dataflow. Hardware properties of **Algorithm** 4 are listed in Tab. 3.12. In order to exploit LDR within an ALU subtask, the fourth row realizes parallel processing with multiple δ ifmap heights. Through a

Algorithm 4 Processing proposed dataflow for matrix multiplications

```
Require: $W, I$
Ensure: $O$
Initialization;
for $k_1 \in [1 : C_\text{i}/\alpha], u_1 \in [1 : C_\text{o}/\beta]$ do
    for $k_2 \in [1 : \alpha], u_2 \in [1 : \beta], y_1 \in [1 : H_\text{o}/\delta]$ do
        for $y_2 \in [1 : \delta]$ in parallel do
            $\mathbf{O}[u_1u_2][y_1y_2] += \mathbf{I}[k_1k_2][y_1y_2] \times \mathbf{W}[u_1u_2][k_1k_2]$;
        end for
    end for
end for
Return $O$
```

Table 3.12: Hardware properties in the processing dataflow for matrix multiplications.

Subtask	N^y	Type	D_x^y	S_x^y
ALU	δ	**W**	δ	δ
		I	*1*	*1*
		O	*1*	*1*
Buf	$H_\text{o}\alpha\beta$	**W**	H_o	H_o
		I	β	β
		O	α	C_i
Off	$C_\text{i}C_\text{o}H_\text{o}$	**W**	H_o	H_o
		I	C_o	C_o
		O	C_i	C_i

processing loop within the on-chip buffer, the architecture increases LDR of ifmaps and ofmaps. The dataflow has scalable α ifmap channels and β ofmap channels to adopt the size of an on-chip buffer. The proposed dataflow requires less on-chip buffer for product terms than TPU, while TPU has better scalability to adapt most processing tasks.

From Tabs. 3.11 and 3.12, we can find out which dataflow is better when processing different matrix multiplications. According to evaluation metrics, the TPU processing dataflow is better to meet most processing tasks since it has three scalable parameters to exploit LDR of all kinds of data in an ALU subtask. The proposed dataflow only has one scalable parameter γ to exploit weight LDR in an ALU subtask.

Once the processing task is decided, the proposed evaluation metrics are able to evaluate the processing dataflow without hardware implementation.

Table 3.13: Energy and area of related hardware components.

Component (16-bit)		Energy / Access [pJ]	Area [μm^2]
Multiplexer		0.01	7
Multiplier		0.21	258
Adder		0.03	31
On-chip SRAM	512 B	1.43*	18801
	8 kB	6.63*	256901
Off-chip SRAM	8 MB	104.45*	–

*: Average energy consumption of read and write.

3.7 Hardware Design Evaluation

The evaluation metrics can predict the number of accesses and access costs of various hardware components in actual DNN processing. We verify the accuracy of the metrics in terms of processing time, energy consumption, and area, by comparing them with results from RTL simulations.

Evaluation Setup

We collect energy and area for logics and registers from a standard cell library in a 65-nm process technology. The energy consumption of one access to an on-chip buffer and an off-chip memory is collected from the SRAM model in CACTI Ver. 7.0 [15] under a 65-nm process with a 1.2 V supply voltage. Tab. 3.13 summarizes energy consumption. All candidates are built in a 16-bit fixed-point representation, since the number of bits in a word is set to 16 bits in a case study. According to Tab. 3.13, we utilize the sum of one multiplier energy and one adder energy to represent one ALU energy. We utilize the access energy of 512 B SRAM to represent the register access cost. We utilize the access energy of 8 kB SRAM to represent the on-chip buffer access cost. We utilize the access energy of 8 MB SRAM to represent the off-chip memory access cost. Therefore, we set energy parameters $k^{\text{ALU}}_{\text{Energy}}$, $k^{\text{Reg}}_{\text{Energy}}$, $k^{\text{Buf}}_{\text{Energy}}$, and $k^{\text{Off}}_{\text{Energy}}$ to 240 fJ, 5.6 fJ/word, 1.6 fJ/word, and 104 pJ, respectively. Similarly, we set area parameters $k^{\text{ALU}}_{\text{Area}}$, $k^{\text{Reg}}_{\text{Area}}$, and $k^{\text{Buf}}_{\text{Area}}$ to 289 μm^2, 4.59 μm^2/bit, and 3.92 μm^2/bit, respectively. We define the time required to read data from registers, make computations in ALUs and write-back registers for the time of one clock cycle, which is represented by $k^{\text{ALU}}_{\text{Time}}$. A 200 MHz clock cycle is assumed. We assume that two clock cycles are required for one access to on-chip buffers and 6 clock cycles for one access to off-chip memories. We thus set time parameters $k^{\text{ALU}}_{\text{Time}}$, $k^{\text{Buf}}_{\text{Time}}$, and $k^{\text{Off}}_{\text{Time}}$ to 5 ns, 10 ns, and 30 ns, respectively.

In the RTL simulation, we model each hardware consisting of multipliers, adders, registers, on-chip buffers as well as an Network-on-Chip (NoC) between registers and

ALUs and an NoC between on-chip buffers and registers. For the NoC which is capable of all-to-all data transfer, a hierarchical mesh network (HM-NoC) in Ref. [27] is assumed. We evaluate the energy and area of the HM-NoC based on the number of multiplexers required for a specified bandwidth based on an HM-NoC model [27]. From the code tracing log of the RTL simulation and the physical data in Tab. 3.13, we evaluate the energy consumption of the target accelerator as,

$$
\begin{aligned}
E =& E_{\mathrm{MUL}} \times N_{\mathrm{MUL}} + E_{\mathrm{ADD}} \times N_{\mathrm{ADD}} \\
& + E_{\mathrm{Reg\_NoC}} \times N_{\mathrm{Reg\_NoC}} + E_{\mathrm{Buf\_NoC}} \times N_{\mathrm{Buf\_NoC}} \\
& + E_{\mathrm{REG}} \times N_{\mathrm{REG}} + E_{\mathrm{BUF}} \times N_{\mathrm{BUF}} + E_{\mathrm{OFF}} \times N_{\mathrm{OFF}},
\end{aligned}
\tag{3.11}
$$

where E is the total consumed energy, E_{MUL} and E_{ADD} are the energy of single multiplier process and single adder process, respectively. $E_{\mathrm{Reg\_NoC}}$, $E_{\mathrm{Buf\_NoC}}$, E_{REG}, E_{BUF}, and E_{OFF} represent the energy of single access to each of the following components: the register-ALU NoC, the buffer-register NoC, registers, on-chip buffers, and off-chip memories. N_{MUL}, N_{ADD}, $N_{\mathrm{Reg\_NoC}}$, $N_{\mathrm{Buf\_NoC}}$, N_{REG}, N_{BUF}, and N_{OFF} are the total number of access to the following components: multipliers, adders, the register-ALU NoC, the buffer-register NoC, registers, on-chip buffers, and off-chip memories.

The area of on-chip logics including NoCs, ALUs, buffers, and registers is evaluated through required hardware resources from RTL simulation under a 65-nm process technology with a 1.2 V supply voltage. On-chip SRAM area is collected from the SRAM model in CACTI Ver. 7.0 [15].

Experimental Results

For the evaluation of the proposed evaluation metrics in different tasks, we have estimated the energy, the processing time, and the area of the accelerators based on the dataflows of the proposed dataflow, DSIP [55], and TPU [24] for all layers in Alexnet [56]. Fig. 3.15 shows the results of the evaluation metrics for the three dataflows. The RTL simulation results of the proposed dataflow are also introduced in the figure. The available buffer capacity is fixed to 256 kB for all dataflows. The number of ALUs available is fixed to 256 for all dataflows. The on-chip bandwidth between buffers and registers is set to 128 bytes/access. The off-chip bandwidth between off-chip memories and on-chip buffers is set to 4 bytes/access. We scale α, β, γ, and δ to adapt the different shapes of layers.

First, we compare the difference in performance evaluation between the evaluation metrics and the RTL simulation, taking the proposed dataflow as an example. A comparison of other dataflows results in a similar result, and thereby omitted in Fig. 3.15 for clarity. The estimated performance by the evaluation metrics basically meets the results of the RTL simulations. The discrepancy in energy consumption is 6.9% on average for each DNN processing in Alexnet. There are three sources for the deviation. First, the end of cycles of each subtask leads to energy loss in the actual hardware implementation, which is not considered in the evaluation metrics. Second, NoCs are not considered in the evaluation metrics. Third, the access cost of ofmaps is different from the cost of ifmaps and weights. The data movement of ifmaps and weights usually requires read access to memories, while the movement of ofmaps requires both read and write accesses. The time metric provides almost identical results with the cycle-base RTL simulation since the metric calculates the number of clock cycles correctly once the dataflow begins. Differences from RTL simulation exist only in the initialization of the processing which consumes negligible clock cycles in the whole processing. The discrepancy in area is 1.7% on average for each DNN processing in Alexnet. A main source of the deviation is the area of NoCs that is not considered in the metric.

Fig. 3.15 also explains Performance-Power-Area analysis based on the evaluation metrics for each layer in Alexnet. The proposed dataflow shows the best energy efficiency among all layers in Alexnet, which achieves 1.2 × energy reduction and 1.2 × time reduction than the DSIP dataflow when processing CONV layers. However, the DSIP dataflow achieves 1.8 × area reduction than the proposed dataflow. The reason is that the proposed dataflow requires to store all ifmap pixels on-chip to reduce off-chip memory access. For the processing in FC layers, the energy consumption in the TPU dataflow and that in the proposed dataflow are almost the same, since the off-chip memory accesses dominates the overall energy consumption.

Fig. 3.16 introduces results based on the evaluation metrics with the proposed dataflow and the DSIP dataflow under different numbers of ALUs when processing CONV3 layer in Alexnet. RTL simulation results of both dataflows are also shown in the figure. The available buffer capacity is fixed to 256 kB for both dataflows. The number of ALUs available is ranged from 16 to 2048. The on-chip bandwidth between buffers and registers is set to 128 bytes/access. The off-chip bandwidth between off-chip memories and on-chip buffers is set to 4 bytes/access.

Proposed evaluation metrics are able to predict hardware performance well. Discrepancies in energy consumption are 5.4% and 1.8% on average with different numbers of ALUs for the proposed dataflow and the DSIP dataflow, respectively. The time metric provides the identical results with the cycle-based RTL simulations. The discrepancy in the area is 1.0% and 1.3% on average for the proposed dataflow and the DSIP dataflow, respectively.

The results of both dataflows are basically consistent with the discussion which analyzes different Performance-Power-Area trade-offs due to dataflow differences. The number of ALUs has a smaller effect on the energy consumption of the proposed dataflow than that of the DSIP dataflow. The reason is that parameter β in the DSIP dataflow affects both the number of ALUs and the on-chip buffer capacity. As a result, if there are less than 256 multipliers, the DSIP dataflow suffers from more off-chip memory accesses with smaller on-chip buffers. On the other hand, the consumed energy of the proposed dataflow does not vary much with respect to the number of ALUs, since LDR and LSR of ifmaps and ofmaps are not affected by the number of ALUs. The processing time decreases as the number of ALUs increases for both dataflows. The area consumption of the DSIP dataflow is smaller than that in the proposed dataflow when the number of ALUs is smaller than 256. The reason is that a smaller on-chip buffer capacity leads to area reduction in the DSIP dataflow.

We are able to utilize the evaluation metrics to explore the best Performance-Power-Area trade-off under hardware constraints. We next show the results of a trade-off analysis, taking the proposed dataflow and CONV3 layer in Alexnet as an example. We examine the effects of on-chip buffer capacity, on-chip buffer bandwidth, and off-chip memory bandwidth. Default values for the number of ALUs, the bandwidth of on-chip buffers, the bandwidth of off-chip memories are 256, 128 bytes/access, and 4 bytes/access, respectively.

Fig. 3.17 shows the evaluation metrics as functions of the on-chip buffer capacity. In order to adopt different capacity limitations, the proposed dataflow scales α and β. The decrease of α and β reduces the access cost to on-chip buffers, while increases the number of off-chip memory accesses. Therefore, the capacity increase of on-chip buffers leads to energy reduction, time reduction and area increase. If we consider a Performance-Power-Area trade-off, the on-chip buffer capacity of 256 kB leads to the minimum product of energy, time, and area. Fig. 3.18 explains the effect of on-chip buffer bandwidth on the processing time. The increase in the bandwidth leads to a decrease in the processing time. However, when the on-chip bandwidth is larger than 128 bytes/access, the data transfer time between on-chip buffers and registers can no longer be reduced since all required data can be transferred by one buffer access. It is appropriate to design hardware with the on-chip buffer bandwidth of 128 bytes/access. Fig. 3.19 shows the processing time as functions of off-chip memory bandwidth. As expected, the off-chip memory bandwidth directly affects the processing time. It is thus requested to make the off-chip memory bandwidth as large as possible.

The key idea in this chapter is to decompose the entire hardware into several independent hardware components and analyze subtasks in each hardware component. This chapter proposes the evaluation metrics based on the local data reuse and the local space reuse in each subtask. Proposed evaluation metrics are dedicated to quantitatively evaluate the cost of data movement under various processing conditions. Simulation results show the good accuracy of the evaluation metrics when processing Alexnet. According to the proposed metrics, we can easily select better processing dataflow under target hardware constraints without complicated hardware verification.

3.8 Summary

In DNN processing, the data movement among memories and processing elements consumes most of time, energy, and area. At the same time, the number of operations in DNN processing is usually much larger than the number of hardware resources. This chapter proposes two hardware properties, local data reuse and local space reuse to describe the quantitative cost of data movement in each memory hierarchy. Based on the properties, this chapter proposes a new set of hardware metrics to analytically estimate energy, throughput, and area from the processing dataflow. This chapter proposes an efficient CNN processing dataflow with a small on-chip memory requirement. Furthermore, according to the analytical evaluation of various processing dataflows and various processing environments, the proposed evaluation metrics effectively evaluate the processing dataflow without hardware implementation. Designers are able to use the evaluation metrics to greatly reduce the workload of hardware.

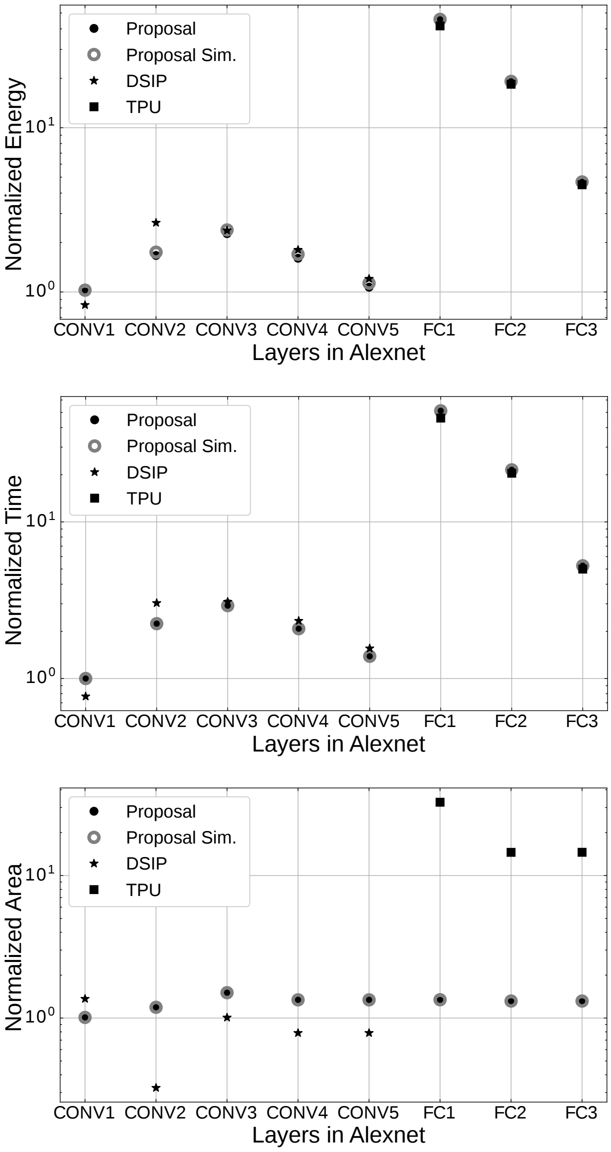


Figure 3.15: Verification for the proposed metrics with various processing tasks in Alexnet. All X axises consist of five CONV layers in Alexnet. Y axises refer to energy consumption, time consumption and area consumption, respectively. All consumption is normalized by the evaluation metric result of the proposed dataflow in CONV1.

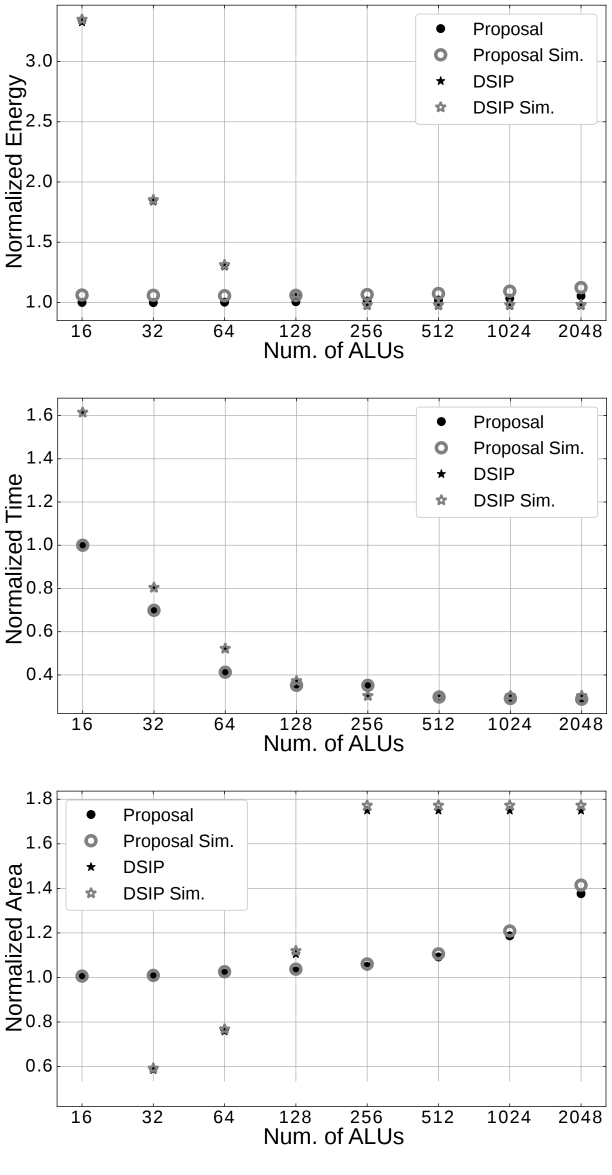


Figure 3.16: Verification for the accuracy of proposed metrics with a different numbers of ALUs. The target processing task is CONV3 layer in Alexnet. X axis shows the number of ALUs available. Y axises refer to energy consumption, time consumption and area consumption, respectively. All consumption is normalized by the evaluation metric result with 16 available ALUs of the proposed dataflow.

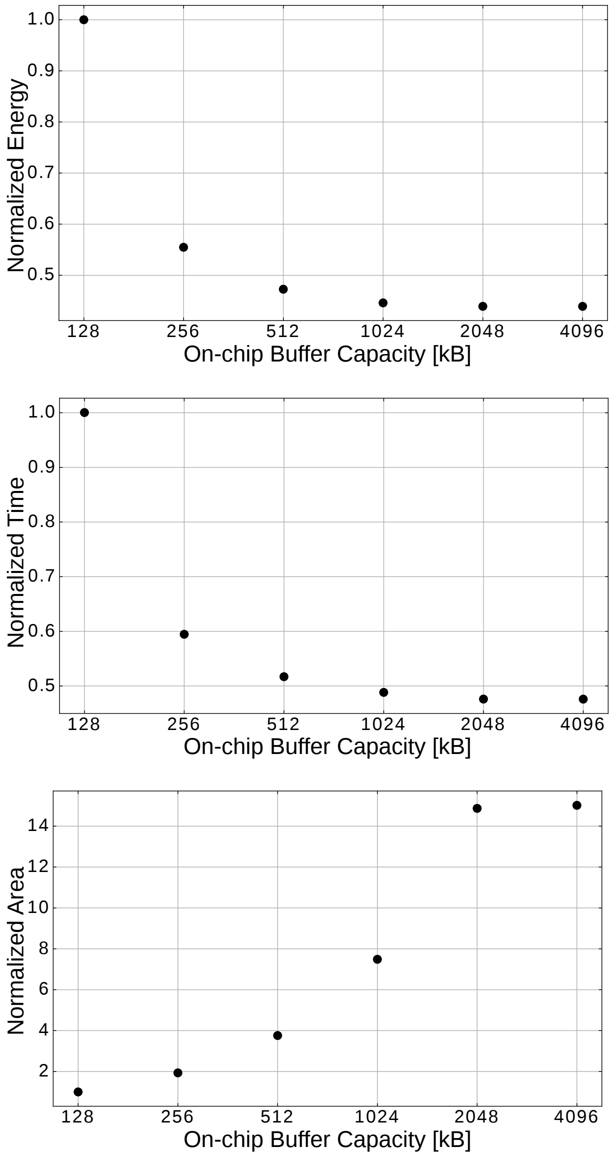


Figure 3.17: Estimated performance for CONV 3 layer in Alexnet under different buffer capacity constraints. X axis shows the on-chip buffer capacity available. Y axises refer to energy consumption, time consumption and area consumption, respectively. All consumption is normalized by the evaluation metric result with 128 kB available buffer capacity.

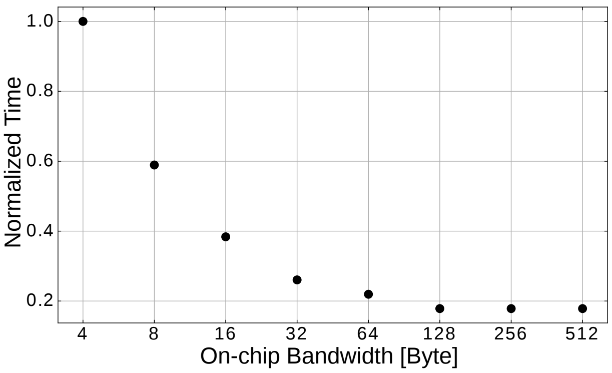


Figure 3.18: Estimated performance for CONV 3 layer in Alexnet under different on-chip bandwidths between buffers and registers. The X axis shows the off-chip bandwidth between off-chip memories and on-chip buffers. The Y axis refers to the processing time. The processing time is normalized by the processing time with an on-chip bandwidth of 4 bytes/access.

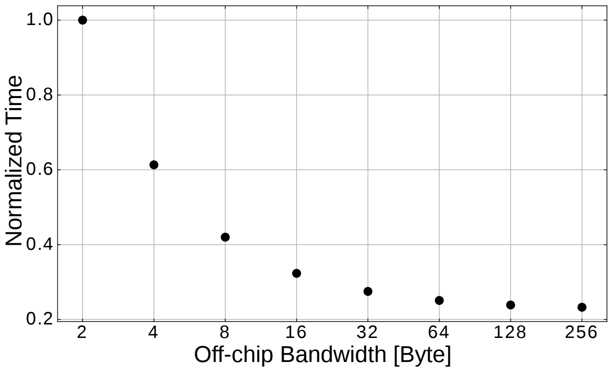


Figure 3.19: Estimated performance for CONV 3 layer in Alexnet under different off-chip bandwidths between off-chip memories and buffers. The X axis shows the off-chip bandwidth between off-chip memories and on-chip buffers. The Y axis refers to the processing time. The processing time is normalized by the processing time with an off-chip bandwidth of 2 bytes/access.

Chapter 4

Sparse DNN Accelerator for Irregular Data Access Pattern

4.1 Introduction

In order to adopt limited computing resources of endpoint devices, many studies utilize pruning techniques to reduce the total amount of parameters in DNNs [61–65]. Since DNN processing often uses Rectified Linear Unit (ReLU), this means that many pixels in feature maps are also be zeroed. As a result, the matrices involved in DNN processing are often sparse [66]. Focusing on the sparsity of DNN processing, previous hardware accelerators transmit only non-zero data to the Processing Elements (PEs), thereby translating the sparsity into energy reduction of the data movement [23, 27, 67]. Once non-zero data are loaded into PEs, hardware implementations are expected to fulfill data into all available Arithmetic Logic Units (ALUs) by index matching [34]. Index matching is to match the coordinate of the weight and the coordinate of the input feature map (ifmap) pixel for each operation to determine whether there is a meaningful operation that produces a non-zero product term. However, the irregular distribution of non-zero data leads to a large matching overhead in parallel processing since accelerators have to match multiple coordinates in parallel [68].

Index matching does not affect parallel-processing performance in dense DNN, since all processing dependencies between ifmaps and weights are clear at the design time. In sparse DNN, index matching becomes critical to decide the parallel-processing hardware complexity due to the data irregularity. The index matching comes to be complicated in the parallel processing task, which should check all loaded weight coordinates and ifmap pixel coordinates with each other to find out meaningful operations that lead to non-zero products. For example, Eyeriss v2 [27] enhances throughput by utilizing multiple scratchpad memories with a pipeline architecture inside PEs to detect meaningful operations. Fetching just right amount of input data with a simple logic to guarantee high PE utilization is important, since the number of non-zero products is unknown in sparse DNN processing. In order to detect meaningful operations quickly in sparse DNN processing, it is required to find out the property to help us understand how many meaningful operations are among ifmap pixels and weights, thereby guiding us to design a simple sparse processing architecture. This chapter views meaningful operations as multiplications that both corresponding ifmap pixels and weights are non-zeros.

To address the challenge of high PE utilization, the third target in this thesis makes the following contributions:

1. This chapter first describes the property called matching type for index matching in DNN processing, which helps to predict the number of meaningful operations efficiently, thereby enhancing PE utilization in sparse DNN processing. This chapter utilizes PE utilization to refer to the average ratio of multipliers that process meaningful operations in the PE.

2. Based on the proposed property, this chapter first introduces a sparse DNN accelerator called Memory Optimized Sparse DNN Accelerator (MOSDA). This chapter makes a hardware evaluation of the proposed sparse accelerator to compare with state-of-the-art DNN accelerators in both dense and sparse DNN processing.

The rest of this chapter is organized as follows. Section 4.2 discusses the motivation and processing properties in sparse DNN processing. Section 4.3 introduces the proposed DNN processing dataflow. Section 4.4 introduces the hardware realization of the proposed processing dataflow. Section 4.5 discusses the simulation results to show the superiority of the proposed hardware realization. Section 4.6 concludes this chapter.

4.2 Background

4.2.1 Sparse DNN Processing

The sparsity of weights and ifmaps can be utilized by gating or skipping operations, thereby improving energy efficiency. In order to further enhance throughput by sparsity, a complex control logic is usually inevitable according to the irregularity of weights and ifmaps in sparse DNN. For achieving high PE utilization, this chapter proposes a method that aims to directly infer the number of operations from the number of loaded non-zero

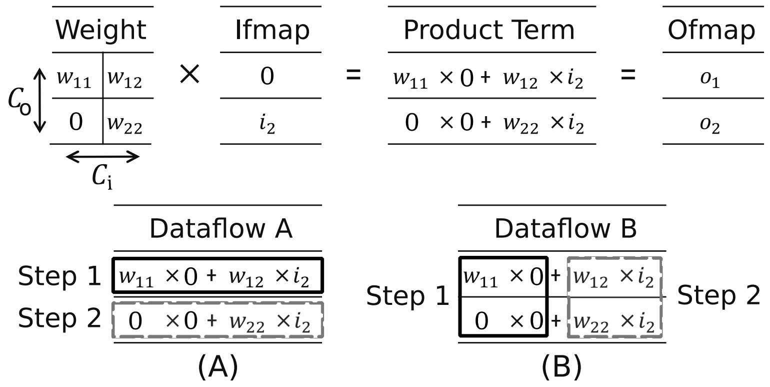


Figure 4.1: Sparse Matrix Multiplication with $C_{\mathrm{i}} = C_{\mathrm{o}} = 2$, $N = H_{\mathrm{w}} = W_{\mathrm{w}} = H_{\mathrm{i}} = W_{\mathrm{i}} = H_{\mathrm{o}} = W_{\mathrm{o}} = 1$. Two possible processing dataflows with 2 multipliers are introduced.

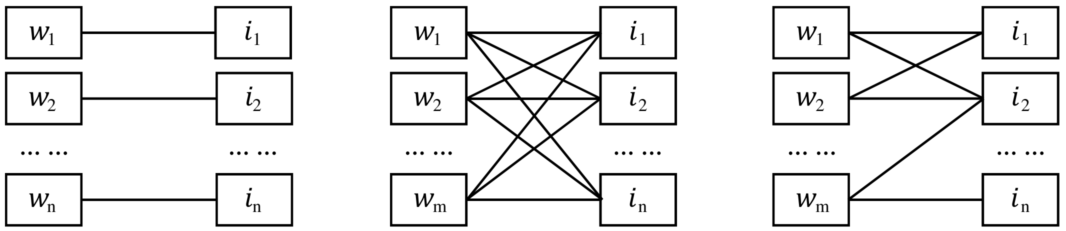
ifmap pixels and the number of loaded non-zero weights so that we can easily arrange weights and ifmap pixels required for massively parallel processing. We fetch just the right amount of input data, thereby maximizing PE utilization with a simple control logic. We focus on the data dependency between ifmaps and weights to evaluate its impact on the number of required operations for sparse DNN.

4.2.2 Motivational Example

In this sub-section, we review a sparse matrix multiplication shown in Fig. 4.1. The shape of the matrix multiplication is $C_{\mathrm{i}} = C_{\mathrm{o}} = 2$. Suppose we have only 2 multipliers available, two possible processing dataflows, Dataflow A and Dataflow B, are shown in Fig. 4.1 (A) and (B), respectively. Dataflow A uses 2 multipliers to generate product terms to the same output feature map (ofmap) pixel in each step. Dataflow B uses 2 multipliers to generate product terms from the same ifmap pixel in each step. According to the definition given in Ref. [4], Dataflow A is an ifmap-stationary dataflow. Dataflow B is an ofmap-stationary dataflow. Both dataflows require two steps to complete the processing when matrices are dense.

For a sparse matrix operation shown in Fig. 4.1, all required data have been loaded into the on-chip buffer, and each buffer can only hold one data for initialization. We observe that the number of multiplications processed in a single step in different dataflows is not the same.

Dataflow A performs two multiplications which have the same C_{o} coordinate in parallel. In the first step, one multiplication is required since the non-zero ifmap pixel i_2 should multiply with one non-zero weight w_{12}, while there are two non-zero weights available. In the second step, one multiplication is required since the non-zero ifmap pixel i_2 should



(a) o-o
One w multiplies with one i.

(b) a-a
All ws multiply with all is.

(c) s-s
Some ws multiply with some is.

Figure 4.2: Three types of processing dependency. Type (a): one w is processed with one i. Type (b): all ws are processed with all i. Type (c): The processing dependency between ws and is is diverse.

multiply with one non-zero weight w_{22}. In Dataflow A, it is difficult to infer the number of multiplications required until we fetch non-zero data into multipliers. It is hard to fulfill all multipliers without additional control logic in Dataflow A.

Dataflow B performs two multiplications which share the C_i coordinate. Since i_1 is zero, there is no required operation in the first step of dataflow B. As a result, the first step can be skipped once we notice i_1 is zero. In the second step, two multiplications between one ifmap pixel i_2 and two weights w_{12}, w_{22} are processed.

In Dataflow B, we can easily infer the number of operations required by the non-zero ifmap amount times non-zero weight amount. As a result, Dataflow B requires less hardware control effort to maximize PE utilization in the case study.

Rather than discussing the overall processing properties of the processing task, this chapter focuses on the processing dependency of the subtask loaded on the hardware. In the next sub-section, we will discuss the impact of processing dependency in subtasks on inferencing the number of operations.

4.2.3 Processing Dependency in DNN

In dense DNN processing, it is simple to infer the number of operations required since the processing dependency between ifmaps and weights is clear at the design time. In sparse DNN, inferring the number of operations becomes critical for the processing speed and the hardware complexity. From the perspective of parallel processing hardware design, the processing dependency can be divided into three categories.

Type (a) $[o - o]$ in Fig. 4.2 refers that one ifmap pixel should be processed with only one weight in the processing task. If there are x non-zero ifmap pixels and y non-zero weights in sparse processing task, we can only infer that there are at most x or y operations required to be processed in the task. Therefore, a $[o - o]$ sparse processing task requires a complex hardware effort to maximize PE utilization.

Type (b) $[a - a]$ in Fig. 4.2 refers that all n ifmap pixels should be processed with all m weights in the processing task. If there are x non-zero ifmap pixels and y non-zero weights, we can easily infer that there are xy operations required to be processed in the task. As a result, the number of operations can be easily inferred if the sparse processing task follows $[a - a]$ type.

Type (c) $[s - s]$ in Fig. 4.2 describes all the other cases of the processing dependency. Type $[s - s]$ refers that there are some ifmap pixels required to be multiplied with some weights in the processing task. The number of operations required for one ifmap pixel (or one weight) varies. As a result, in a sparse $[s - s]$ processing task, it is difficult to infer the overall required operations just by the number of non-zero data. Therefore, a $[s - s]$ sparse processing task requires dedicated hardware to maximize PE utilization.

We refer the processing dependency of weights and ifmap pixels in a certain processing task as *matching type*. In Dataflow A of Fig. 4.1, each ifmap pixel multiplies with only the matched weight in each subtask, which means that the matching type in the subtask of Dataflow A is $[o - o]$. In Dataflow B of Fig. 4.1, all ifmap pixels are multiplied with all weights in each subtask which means the matching type in the subtask of Dataflow B is $[a - a]$. Therefore, the hardware implementation of Dataflow A requires special hardware to maximize PE utilization without redundant data movement. On the other hand, the processing with [s-s] matching type can be found, for example, in the parallel processing with 4 multipliers. The parallel processing task of all four multiplications simultaneously in Fig. 4.1 belongs to the [s-s] matching type, which could lead to a complex index matching effort.

In $[a - a]$ matching type, the number of required operations in the processing task can be simply expressed by the number of non-zero ifmap pixels times the number of non-zero weights regardless of sparsity. Therefore, we are able to fetch just the right amount of input data to enhance PE utilization with a simple control logic. As a result, keeping the matching type of the PE subtask being $[a - a]$ is the key to enhance PE utilization.

If we generalize the discussion from matrix multiplications to operations in DNN, matching types in different subtasks are summarized in Tab. 4.1. For convolution operations, the matching type of the corner ifmap pixels changes according to the chosen padding type. Since the central ifmap pixels usually dominate in number, we utilize matching type of the central ifmap pixels to represent the matching type of all ifmap pixels. Therefore, we view the matching type of convolution between ifmap pixels and weights as $[a - a]$. Available $[a - a]$ matching type refers to simple hardware workloads

Table 4.1: Matching types of typical subtasks in DNN.

Subtask Processing Direction	Matching Type in Weights-to-Ifmap Pixels
N	$[a-a]$
C_i	$[o-o]$
C_o	$[a-a]$
H_w	$[a-a]$
W_w	$[a-a]$
H_o	$[a-a]$
W_o	$[a-a]$
D	$[o-o]$
$(H_\mathrm{o}, H_\mathrm{w})$	$[a-a]$
$(C_\mathrm{i}, C_\mathrm{o})$	$[s-s]$

to process the subtask in parallel. In actual hardware accelerators, the chosen subtask for PEs is usually the combination of multi dimensions to maintain the throughput of the accelerators. In the case where the subtask contains both $[a-a]$ subtask and $[o-o]$ subtask, the matching type would be $[s-s]$. Once the subtask for DNN processing is determined, the matching type of the target subtask is fixed and will not change with sparsity.

4.2.4 Related Work

Tab. 4.2 introduces several typical state-of-the-art DNN accelerators. Different accelerators utilize different subtask processing strategies. This also leads to a difference in data-fetching strategies. The subtask adapted by Eyeriss v2 [27] loads data into registers in a PE as much as possible. Although this increases the theoretically available data reuse in PE subtask, it also makes its control logic complicated due to its $[s-s]$ matching type which requires a multi-stage pipeline architecture for each PE to enhance PE utilization. EIE [23], and Cnvlutin [28] have the processing dimension of $(C_\mathrm{i}, C_\mathrm{o})$ in a PE, which leads to the $[s-s]$ matching type. OuterSPACE [34] utilizes outer product to replace the conventional inner product sequence to process matrix multiplications while keeping the matching type as $[a-a]$ in $(C_\mathrm{o}, H_\mathrm{o})$ processing dimension. As a result, OuterSPACE achieves fast and efficient data scheduling. However, Outer SPACE can only handle matrix multiplication, but not other processing targets, such as convolution. In the next section, this chapter proposes a sparse DNN processing dataflow that fully considers the processing properties based on the dataflow in Chapter 3 derived for dense DNN processing. The proposed sparse processing dataflow shows efficiency when facing different processing targets, which are convolutions (CONVs), fully-connected layers (FCs), and Matrix

Table 4.2: Processing dimension of subtasks in typical accelerators.

Accelerator	Target	Subtask Processing Direction	Matching Type
Eyeriss v2 [27]	Sparse CNN	$(C_\mathrm{o}, C_\mathrm{i}, H_\mathrm{o}, W_\mathrm{o}, H_\mathrm{w}, W_\mathrm{w}, D)$	$[s-s]$
EIE [23]	Sparse FC	$(C_\mathrm{i}, C_\mathrm{o})$	$[s-s]$
Cnvlutin [28]	Sparse CONV	$(C_\mathrm{i}, C_\mathrm{o})$	$[s-s]$
OuterSPACE [34]	Sparse Matrix	$(C_\mathrm{o}, H_\mathrm{o})$	$[a-a]$

Multiplications, from the experiment result.

4.3 Sparse DNN Processing Dataflow

In this section, this chapter proposes a sparse DNN processing dataflow based on the matching type discussed in the previous section. The key idea of the proposed dataflow is to ensure that the matching type of the subtasks loaded into PEs is always $[a-a]$.

One batch of the convolution operation requires $C_\mathrm{i} \times C_\mathrm{o} \times H_\mathrm{o} \times W_\mathrm{o} \times H_\mathrm{w} \times W_\mathrm{w}$ multiplications in total. Additions of the same order of magnitude are also required for multiplication results. As shown in Fig. 3.1, each ofmap pixel contains $C_\mathrm{i} \times H_\mathrm{w} \times W_\mathrm{w}$ product terms, each product term needs to occupy one memory space in sequential processing. If we utilize fully parallel processing, we are able to merge $C_\mathrm{i} \times H_\mathrm{w} \times W_\mathrm{w}$ product terms into one partial sum (psum) before accessing the memory to reduce the memory capacity required by the accelerator. The previous work refers to the processing property as *space reuse*. Hence, this chapter notices that there are three processing properties that affect the hardware performance in DNN processing, which are the *data reuse*, the *space reuse* and the *matching type*.

In order to exploit three processing properties, Fig. 4.3 introduces two requirements for the sparse DNN processing dataflow. First, the processing dataflow for sparse DNN processing should guarantee all subtasks in the PE should be with the matching type $[a-a]$. Second, in order to further reduce the data movement in the hardware implementation for the dataflow, data reuse and space reuse should also be exploited as much as possible to reduce the number of memory accesses and memory capacity.

The dataflow in Chapter 3 for dense DNN processing satisfies both two requirements. In the dataflow, the subtask stored in the PE is 2D CONV, where matching type is $[a-a]$. In each clock cycle, the registers fetch multiple ifmap pixels in the same column to multiply with all weights in 2D CONV to exploit data reuse of ifmaps and weights. If we keep the subtask scope within 2D CONV, there are also product terms in the subtask that can be

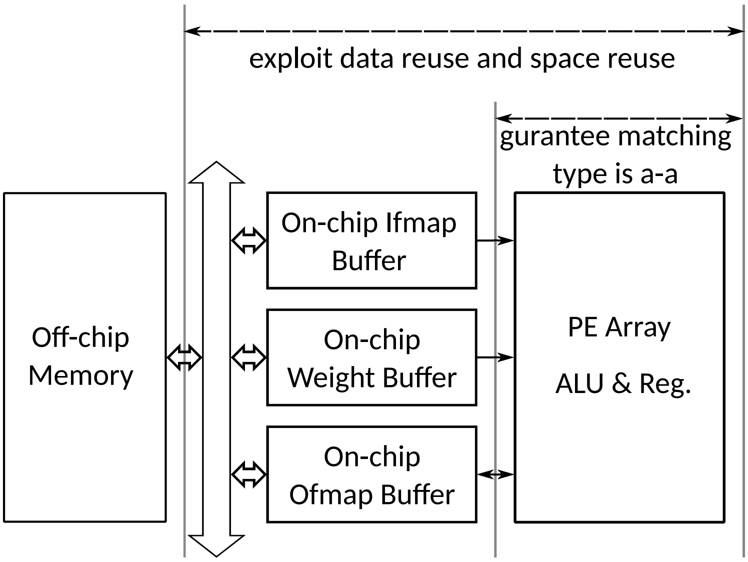


Figure 4.3: Requirements of sparse DNN processing dataflow. First, processing dataflow should guarantee the matching type of the subtask in the PE is $[a - a]$ for sparse DNN processing. Second, the dataflow should exploit data reuse and space reuse as much as possible.

accumulated. In other words, there is also space reuse available in 2D CONV. Therefore, related product terms are added to reduce the required capacity of the on-chip memories.

This chapter first introduces a sparse DNN processing dataflow as an expansion of the dense dataflow in Chapter 3. The proposed sparse processing dataflow does parallel processing within the $(C_{\mathrm{o}}, H_{\mathrm{o}}, W_{\mathrm{o}}, H_{\mathrm{w}}, W_{\mathrm{w}})$ dimensions to guarantee the matching type as $[a - a]$ in the PE. Under the target dimension, we further scale the processing range in the $(C_{\mathrm{o}}, H_{\mathrm{o}}, W_{\mathrm{o}})$ dimensions to match various sizes of processing tasks with a fixed number of multipliers inside a PE. The processing dataflow guarantees that the subtask in the PE is always $[a - a]$ matching type.

Fig. 4.4 introduces a processing example in one PE with 8 multipliers following the proposed processing dataflow. The target convolution consists of three 3×3 weight matrices for 3 different ofmap channels and a 4×4 ifmap matrix. Different ofmap channels contain different number of non-zero weights a, b, c, and d. First, all non-zero weights, ifmap pixels and the index coordinates in 2D CONV are stored in the COO format [69]. In addition, we store the ofmap channel coordinates I, II, and III in the weight register. The 8 multipliers are arranged as 2×4, where each set of 4 multipliers in the same row shares the same weight and each set of 2 multipliers in the same column shares the same ifmap pixel. Two weights are combined into one weight group (WG) and four ifmap pixels are combined into one ifmap group (IG). Once the data are loaded into registers, MOSDA fetches two weights in one WG and 4 non-zero ifmap pixels in one IG to fulfill all 8 multipliers per clock cycle. We adopt ifmap stationary here to exploit data reuse and space reuse. As shown in the right bottom of Fig. 4.4, (WG0 / IG0), (WG1 / IG0), (WG0 / IG1), and (WG1 / IG1) are processed sequentially.

When the product term is generated, we also compute the index information of the product term in the COO format. The ofmap channel of the product term is the same as the ofmap channel of the weight. Product terms in the same row share the same ofmap channel coordinate. We add product terms according to indexes of product terms in the spatial addition stage. By utilizing indexes of product terms, the spatial addition part shown in the right bottom of Fig. 4.4 merges product terms with the same indexes through a crossbar. For example, since there are two product terms that have the same index (1,1) in the first step (WG0 / IG0), two product terms are merged into one under the spatial addition stage. We store psum results from the spatial addition stage into registers and wait for the temporal addition. Psum results stored in registers are not always necessary for the following accumulation. For example, corner product terms with the index $(-1,1)$ in Fig. 4.4 is unnecessary because the index coordinates are out of the range of the output matrix. In order to avoid the energy loss by useless multiplications, the registers at the beginning of the temporal addition stage shield the useless product terms. The temporal addition part shown in the right above of Fig. 4.4 aims to further accumulate psums that can be accumulated between different clocks according to crossbars and the psum register.

Through the proposed dataflow, MOSDA can utilize the potential data reuse and space reuse of 2D CONV while ensuring the matching type as $[a-a]$. Therefore, the method can ensure the PE utilization while reducing the required memory access and memory capacity.

We first propose a DNN hardware evaluation model by local data reuse and local space reuse. We further propose MOSDA exploiting the matching type for sparse DNN processing.

4.4 MOSDA Architecture Based on Matching Type

Architecture Overview

A MOSDA architecture overview is shown in Fig. 4.5. All ifmap pixels and weights are stored in the off-chip memory. On-chip buffers store data which can be accessed in the near future to exploit data reuse and space reuse.

MOSDA architecture can be scaled across a number of dimensions. Table 4.3 introduces key parameters of MOSDA in this chapter. MOSDA consists of 16 PEs, each with

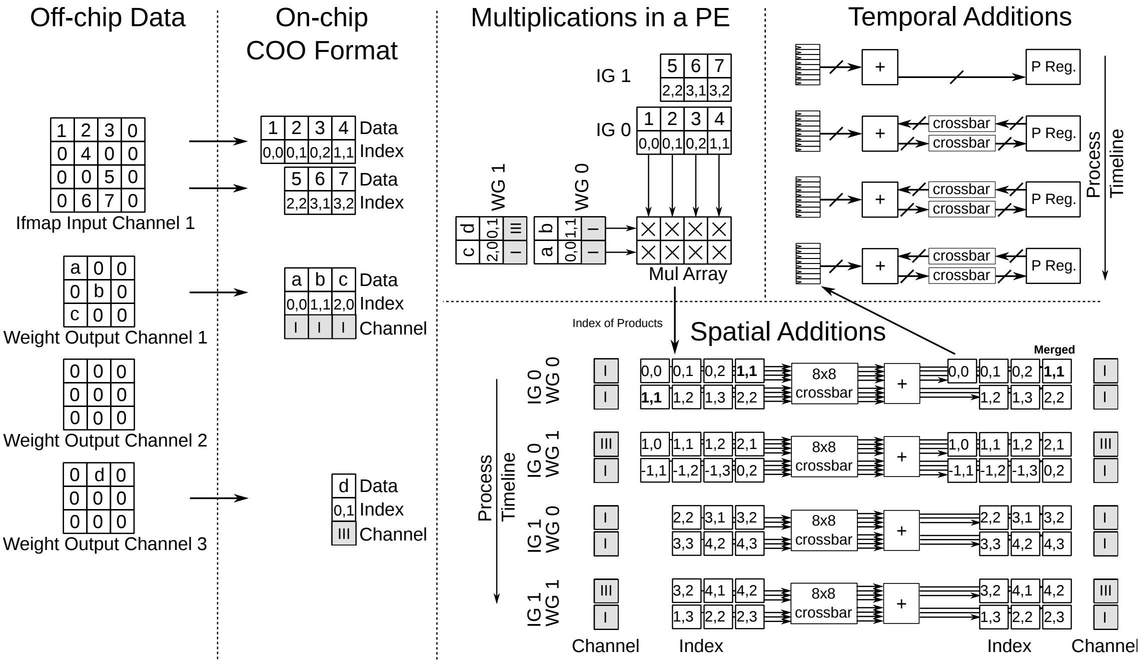


Figure 4.4: The subtasks loaded into the registers always belong to $[a - a]$ matching type. Only non-zero ifmap pixels and weights are loaded into a PE.

a 16 multiplier array. 16 multipliers in one PE can be arranged as 16 rows × 1 column for processing of FC layers or 2 rows × 8 columns for other cases. Each row shares the same weight, and each column shares the same ifmap pixel. We store ifmap pixels and weights in 8 bit. Product terms generated by multiplications are accumulated in 24 bit. When the accumulation is finished, 24-bit psums are converted back to 8-bit ofmap pixels and sent off-chip without coordinate information. We do not use the COO format to store data in the off-chip memory, since MOSDA is also intended for dense DNN processing. Memories carry an 8-bit overhead to encode the coordinate information for each value, which are able to process operations with 16 × 16 matrix in parallel. In the setup, if the sparsity is less than 50%, the data stored under the COO format will take more storage space than the data stored without the COO format. All data are transmitted in the COO format on-chip in this chapter.

Parallel Processing in one PE

We exploit data reuse and space reuse, thereby reducing the cost of data movement within the PE. We utilize a three-stage pipeline architecture to reduce the critical path of the PE.

The main function of the first stage in the pipeline is parallel data fetching and multiplications. In order to exploit the data reuse and space reuse as much as possible, the dataflow further processes DNN ifmap channel by ifmap channel in the C_i dimension as

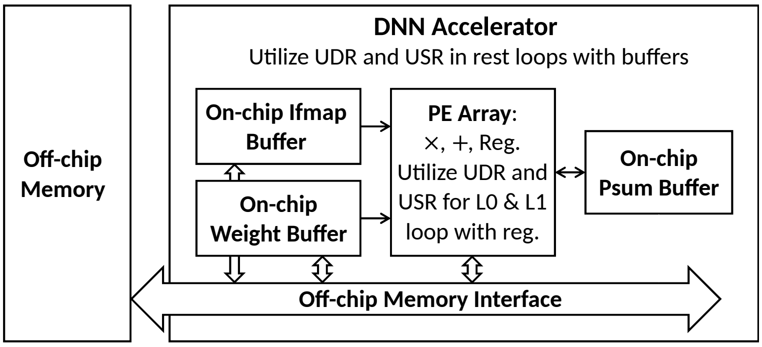

DNN Accelerator
Utilize UDR and USR in rest loops with buffers
Off-chip Memory
On-chip Ifmap Buffer
On-chip Weight Buffer
PE Array:
×, +, Reg.
Utilize UDR and USR for L0 & L1 loop with reg.
On-chip Psum Buffer
Off-chip Memory Interface

Figure 4.5: MOSDA architecture overview. The subtasks loaded into the registers always belong to $[a - a]$ matching type. All PEs share one global buffer for all kinds of data.

Table 4.3: MOSDA Design Parameters.

MOSDA Parameter	**Value**
# PEs	16
Global Buffer	192 KB
PE Parameter	**Value**
# Multipliers	16
Multiplier Width	8 bits
Accumulation Width	24 bits
Weight FIFO in one PE	32x16 bits
Ifmap FIFO in one PE	64x16 bits
Psum Register in one PE	64x32 bits

an outer loop. In order to guarantee the matching type as [a-a], the dataflow processes the task ofmap channel by ofmap channel in the C_o dimension as an inner loop. If non-zero data inside one channel is not enough to fulfill multipliers, MOSDA will fetch data from multiple ofmap channels into one PE to maintain high PE utilization. Fig. 4.4 is a case study that loads data for 3 ofmap channels, since the number of operations is less than the number of multipliers within one ofmap channel. This ensures that PE utilization is not degraded because enough data is loaded into the PE.

The data transmission logic is introduced in Fig. 4.5. For parallel processing of FC layers, 16 multipliers are configured into 16×1. All 16 multipliers share the same ifmap pixel, and each multiplier has a unique weight. For parallel processing of other cases such as convolutions, multipliers are configured into 2×8. 2 multipliers in one same column share the same ifmap pixel, and 8 multipliers in one same row share the same weight. Once multiplications are finished, MOSDA requires a crossbar logic to transmit psums from multipliers to spatial adders.

If the number of non-zero weights in 2D CONV is too small, the required memory bandwidth of the ifmap buffer will be large at every clock cycle. In order to balance the on-chip memory bandwidth of weights and ifmap pixels, if there are too few non-zero weights under a single channel, MOSDA scales parallel processing range of the ofmap channel to multiple ofmap channels to balance memory bandwidth requirement while maximizing the PE utilization.

One of the key issues in sparse DNN processing architecture is high area/power overhead of accumulation due to irregular output data [35, 70]. In this chapter, we reduce the psum overhead by reducing psum register access by reserving a spatial redundant addition network. Fig. 4.5 introduces the psum accumulation logic in MOSDA, which is decomposed into a spatial addition stage and a temporal accumulation stage. The spatial addition stage aims to merge possible product terms into one psum within one clock cycle. We compare the indexes of all product terms with each other in the spatial addition stage, and merge corresponding product terms with the same index. The temporal accumulation stage further accumulates psums among different clock cycles. Both stages are controlled by coordinate information in index logics.

We merge the psums corresponding to the same ofmap pixel under the same step through a spatial addition network, thereby reducing the required psum register accesses. Due to the irregular output issue, a 16 × 16 crossbar is required between 16 product terms and 16 adders.

In the next temporal accumulation stage, we write merged psums into psum registers according to the coordinate index. Through the coordinate from the spatial addition stage, first we read the corresponding psums in the registers. Next, we add the read psums to the psums from the spatial addition stage and store them in psum registers. Once all the operations in the current ofmap channel have been completed, we write the psums back to the global buffer. Considering the worst case, we need to write 16 psums into 64 psum registers at the same time. Although this will cause a large area overhead, it ensures the MOSDA processing speed.

In order to limit register accesses by generated psums, we make the spatial addition and send psum results to psum registers. Considering the space reuse existed under different clock cycles, we continue to do the temporal accumulation for product terms stored in psum registers to reduce the required accesses to the global buffer.

Data Transmission among PEs

Due to the scale limitation of the all-to-all transmission network, the psum register capacity cannot be too large, thus limiting exploiting the data reuse in a single PE. We ensure that psum results of different PEs do not need to undergo further accumulation operations since they belong to different ofmap channels. The number of ALUs for a

Table 4.4: MOSDA Specifications.

Technology	65nm SOTB
Core Area	2mm × 6mm
Gate Count (Logic Only)	3066k gates (NAND-2)
On-chip Memory	192 KB SRAM + 6 KB Register
Clock Frequency	200 MHz
Peak Throughput	153.6 GOPS
Filter Size	All
Ifmap Size	All
Processing Type	CONV, and FC (Matrix Multiplication)

single PE is limited by its crossbar structure and cannot be large. MOSDA can increase the number of PEs for achieving higher throughput. As the number of PEs increases, throughput and power consumption will increase linearly. We use 16 PEs in this chapter as a case study. Different PEs are responsible for processing calculation tasks of different ofmap channels, thereby improving overall processing speed. Please note that a single PE may be responsible for multiple ofmap channels if the number of operations corresponding to a single ofmap channel is small compared to the number of multipliers in a PE, which ensures to maintain high PE utilization.

The global buffer provides the required non-zero weights and ifmap pixels to each PE sequentially. After data fetching finishes, PEs do processing in parallel. Once processing tasks in PEs are completed, each PE sequentially restores their respective psum results back to the global buffer.

MOSDA is able to switch the data transmission mode from sparse mode to dense mode to save energy overhead due to coordinate information. When MOSDA is initialized, information such as the size of the processing task and whether it is sparse is stored in the on-chip logic control. In the dense mode, MOSDA no longer activates the COO format converter or stores the coordinate information of ifmap pixels and weights in the global buffer.

4.5 Experimental Results

In this section, we verify the efficiency of MOSDA with 16 PEs by post-layout cycle-accurate gate-level simulations under a 65-nm Silicon-on-Thin-Box (SOTB) process with a 1.2 V supply voltage. The bandwidth between the global buffer and PEs transmit 48 words once for weights and ifmap pixels, or 24 words once for psums. The specifications are summarized in Tab. 4.4. The peak throughput of MOSDA is defined as the total throughput of processing elements, which consists of 256 multipliers and 512 adders in total. Tab. 4.5 summarizes the energy of major logic components in a 1.2 V supply voltage. All energy values except for on-chip SRAM are based on the average values for one operation during overall post-layout simulations. All hardware components are built in the fixed-point representation. More specifically, the energy of multipliers is collected by one multiplier, which has 8-bit inputs and one 16-bit output. The energy of adders is collected by one 16-bit adder for psum accumulations. The energy of the crossbars is evaluated by transmitting one 16-bit data through crossbars. The energy consumption of the access to the global buffer is collected from SRAM model in CACTI Ver. 7.0 [15] under a 65-nm process with a 1.2 V supply voltage.

MOSDA logic part is evaluated from the post-layout simulations. In order to verify the validity of the energy analysis through simulations and make a fair comparison with other state-of-the-art accelerators, we further evaluate synthesized MOSDA with sparse Alexnet. We did a gate-level simulation using NC-verilog to generate Value Change Dump (VCD) file. We further convert VCD to toggle information, which is dumped to Switching Activity Interchange Format (SAIF). Finally, the energy consumption is estimated from the number of toggles. The data transmission Network-on-Chip (NoC) consists of transmission logics between weight FIFO and multipliers, logics between ifmap FIFO and multipliers, and a crossbar between multipliers and spatial adders. The area consumed by the crossbar is 85% of the total NoC area. The area of the psum register consists of both register cells and the crossbars to fetch data into registers. The energy and area of MOSDA buffer part are collected from CACTI model. Based on the code tracing logs, we calculate the access activities of the global buffer. Combining the activities and the physical SRAM data in Tab. 4.5, this chapter simulates the energy consumption of the overall MOSDA performance.

MOSDA core area is 12 mm^2, which consists of a 192 KB on-chip SRAM buffer and the 16 PEs. The overall logic gate count is 3066k NAND-2 gates. Fig. 4.6 (A) introduces the area breakdown in MOSDA. The global buffer accounts for 46.6% of the total core area. The control logic for data transmission between the global buffer and PEs consumes 3.7% of the total core area. The COO format convertor is included in the area breakdown of the control logic. The PE composed of multipliers and registers accounts for 49.7% of the total core area. Fig. 4.6 (B) introduces the reason for PE area overhead. In order to ensure all generated product-terms stored in registers in time, the psum register inside the PE requires 16 read-write ports, which consume 42.8% of the PE area. The accumulation overhead is the cost to handle the irregular output of sparse DNN processing. Simultaneously, the NoC (Crossbars) requires 22.1% of the PE area and

Table 4.5: Related energy consumption in convolution operations.

Component	Energy / Access [pJ]
8-bit Multiplier (E_{MUL})	0.14
24-bit Adder (E_{ADD})	0.02
16×16 Crossbar (E_{NoC})	0.07
8-bit Register (E_{REG})	0.09*
On-chip SRAM (E_{BUF}) 8 kB	3.31*

*: Average energy consumption of read and write.

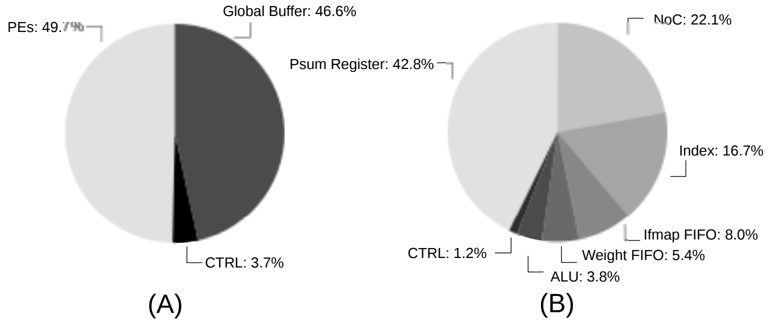


Figure 4.6: (A) MOSDA area breakdown. (B) Area breakdown in one PE.

the index logic requires 16.7%. Both parts require a complex data transmission logic to handle the unpredictable product terms generated from multipliers.

We simulate the energy consumption of MOSDA under various benchmarks including convolution layers, FC layers, and matrix multiplications. The benchmarks are introduced in Tab. 4.6, which consist of convolution layers and FC layers in Alexnet [56], sparse Alexnet [65], MobileNet [53] with width multiplier of 1.0 and input size of 224 × 224, corresponding sparse Mobilenet with data from ImageNet dataset [71], and two matrix multiplications by multiplying with itself in SNAP [72]. The stride U of conv1 in Alexnet is 4. As a result, the matching type between weights and ifmap pixels in conv1 comes to be $[s - s]$ since not all weights are required to multiply with all ifmap pixels. To solve this issue, Fig. 4.7 introduces a decomposition method to convert the CONV into multiple smaller CONVs. Small CONVs are all $[a - a]$ matching types except the corner data. According to the decomposition method, we are able to process convolution layer with stride more than 1 with $[a - a]$ matching type.

Tab. 4.6 also introduces the average PE utilization in various benchmarks. PE utilization is the average ratio of multipliers that handle meaningful multiplications when PE is required to process operations. Note that the active rate of multipliers under data transmission time among memories is not taken into account. The key idea of MOSDA is to keep the matching type in the PE subtask always as $[a - a]$. The corner data in

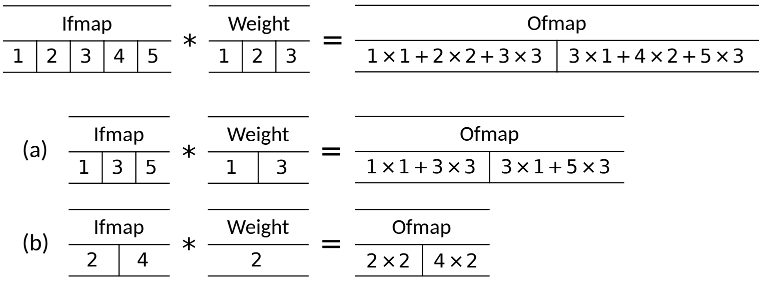


Figure 4.7: Decomposition method of 1-dimensional convolution when stride is larger than 1: In a case study, sizes of the ifmap pixel, the weight and the stride are 5, 3, and 2, respectively. When the stride is 2, the matching type will be $[s - s]$. Therefore, the convolution is decomposed into two smaller convolutions (a) and (b) with matching types almost $[a - a]$.

Table 4.6: Multiplier array utilization of MOSDA benchmarked with Alexnet, and MobilNet that has batch size of 1 and matrix multiplications.

Target	Non-zero Weight	Non-zero Weight Ratio	Non-zero Ifmap Ratio	Off-chip Memory Access [MB/feature]	Average Mul Utilization
CONV Layers 1-5 in Alexnet [56]	2.2 M	-	-	10.0	99.33%
FC Layers 1-3 in Alexnet [56]	58.6 M	-	-	58.7	99.84%
CONV Layers 1-5 in Sparse Alexnet [65]	0.2 M	7.7%	69.8%	4.3	90.01%
FC Layers 1-3 in Sparse Alexnet [65]	5.9 M	10.1%	57.1%	5.9	99.96%
CONV Layers 1-27 in MobileNet [53]	4.1 MB	-	-	5.0	99.05%
FC Layer 1 in MobileNet [53]	1.0 MB	-	-	1	97.71%
CONV Layers 1-27 in Sparse MobileNet [73]	0.4 MB	10.0%	47.7%	0.5	92.34%
FC Layer 1 in Sparse MobileNet [73]	0.1 MB	10.0%	49.8%	0.1	99.4%
Wiki-Vote [72]	103.7 KB	0.2%	0.2%	1.5	99.99%
Facebook [72]	88.2 KB	0.5%	0.5%	1.2	99.99%

convolution processing does not have the matching type [a-a], which is the main reason for the PE utilization loss. As a result, the PE utilization of FC layer in Alexnet and matrix multiplications are as high as 99.96% and 99.99%, respectively. In CONV processing, although we assume that the matching type in convolution processing is $[a - a]$, the actual situation is that the corner data of the ifmaps in the convolution operation does not need to be multiplied by all weights. Therefore, this assumption led to a slight decrease of PE utilization in the sparse convolution processing, but still maintained at 90.01% and 92.34% for sparse Alexnet and sparse MobileNet, respectively.

Following the matching type of $[a - a]$, MOSDA keeps PEs always busy without redundant data fetching. Fig. 4.8 (a) shows the throughput comparison between MOSDA architecture from the dense mode to the sparse mode and a state-of-the-art dense processing architecture from Chapter 3. Benchmarks include Alexnet and Matrix Multiplications. The time consumption consists of processing time by multipliers, and data transmission time from the global buffer to registers in PEs. In order to make a fair comparison, both architectures consist of the same number of multipliers and the same on-chip bandwidth

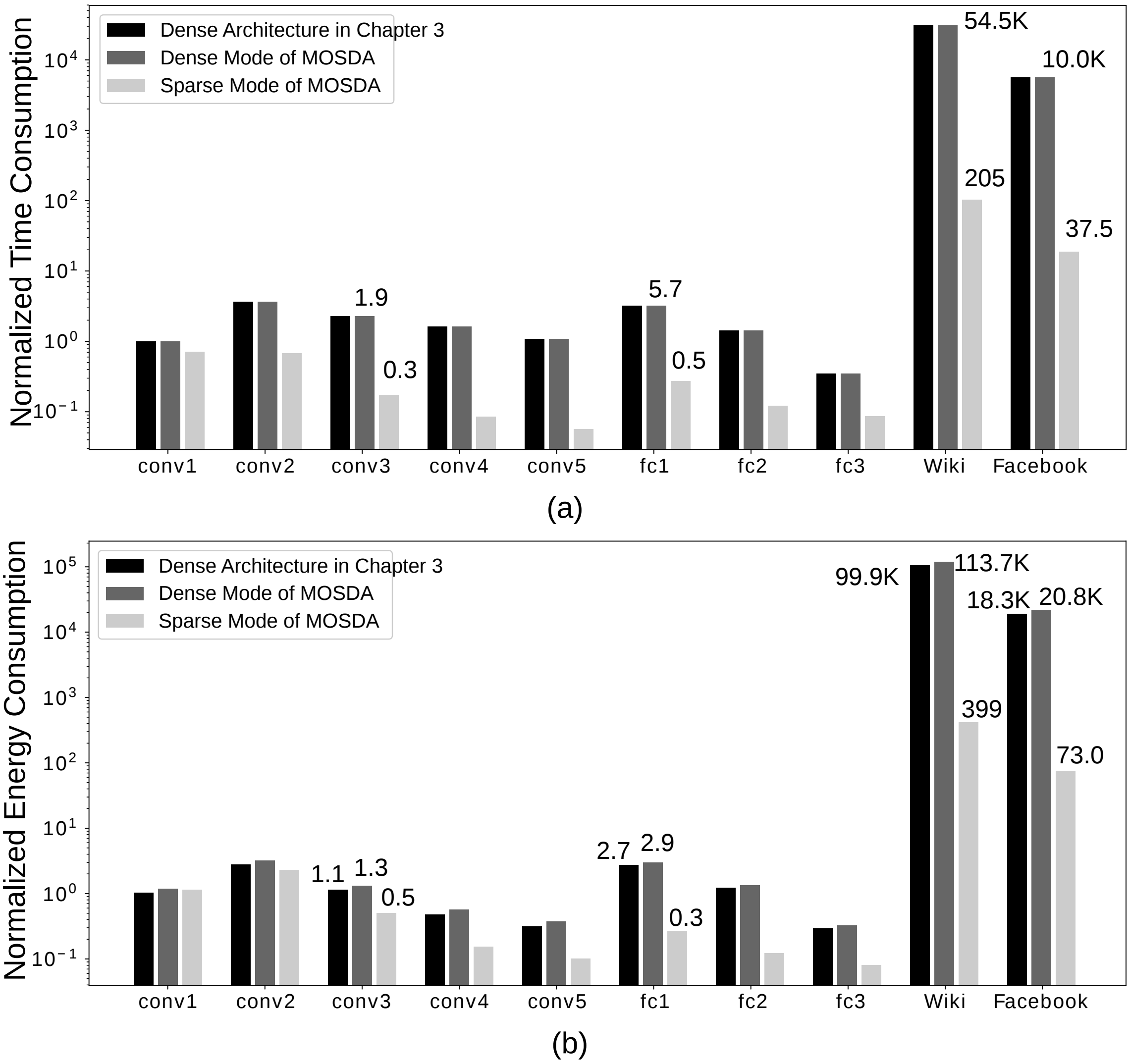


Figure 4.8: (a) Normalized time consumption in dense and sparse benchmarks including Alexnet and Matrix Multiplications. (b) Normalized energy consumption in dense and sparse benchmarks including Alexnet and Matrix Multiplications.

between the global buffer and PEs. Meanwhile, the sparse mode achieves 7.0×, 11.4×, and 265.9× less time consumption than the dense mode in Alexnet conv3, Alexnet FC1 and Wiki-Vote Matrix Multiplication, respectively. The processing time is dramatically decreased with high PE utilization. Among all layers in dense Alexnet, data transmission between the global buffer and PEs consumes 38.1% of the on-chip time consumption. As we expected in sparse Alexnet, data transmission between the global buffer and PEs comes to be 62.2% of the on-chip time consumption, which infers that the time consumed by data transmission between the global buffer and PEs is hard to be reduced despite matching types inside the PE. The sparse mode achieves 5.3× better overall time reduction against the dense mode when processing Alexnet. The throughput of MOSDA dense mode is the

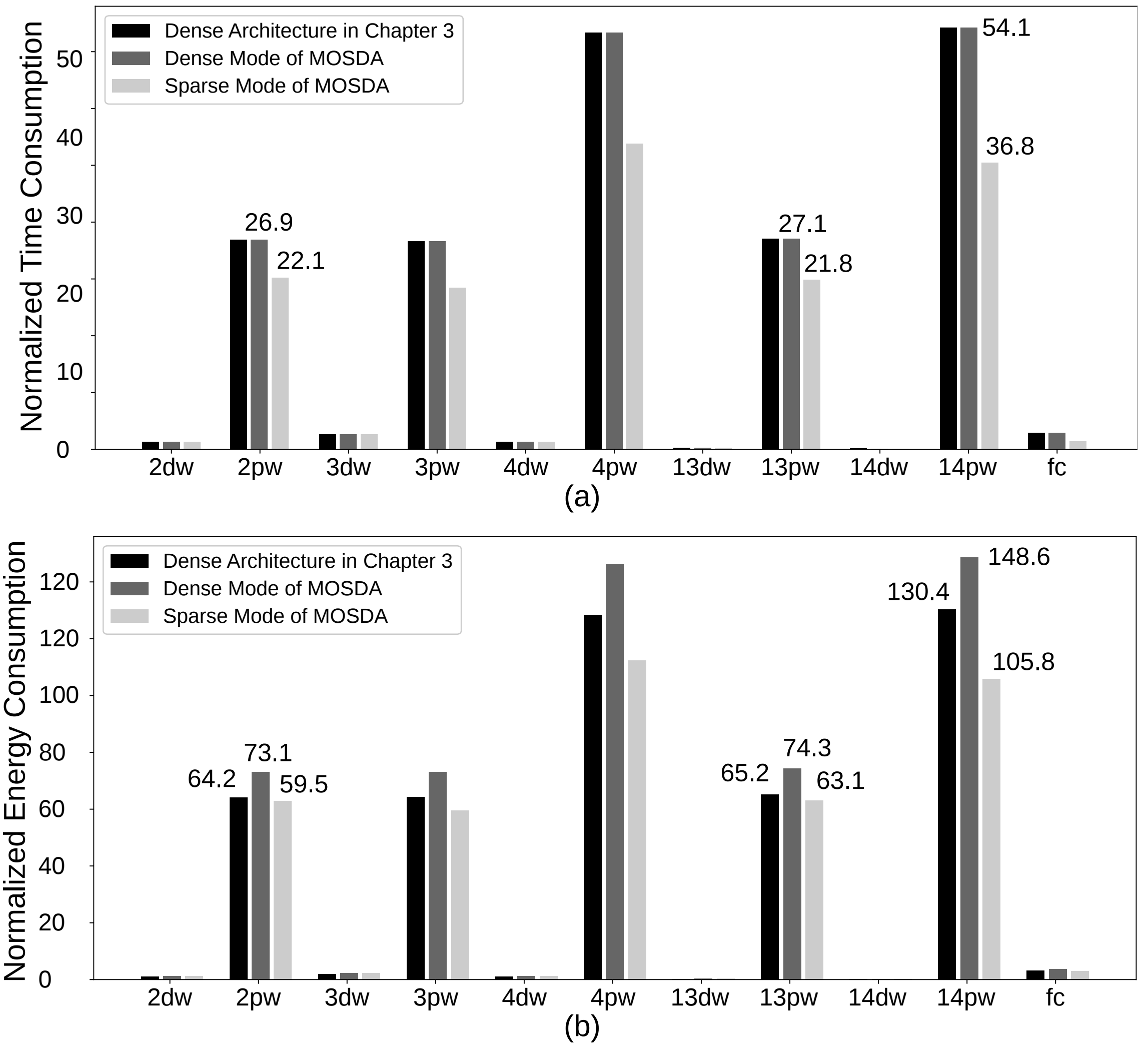


Figure 4.9: (a) Normalized time consumption in dense and sparse benchmarks including typical layers in MobileNet. (b) Normalized energy consumption in dense and sparse benchmarks including typical layers in MobileNet.

same as the architecture in Chapter 3, since the number of multipliers and the on-chip bandwidth is set to be the same.

Fig. 4.8 (b) introduces the energy efficiency under various benchmarks. By considering sparsity, energy consumption is also reduced. The sparse mode achieves 2.6×, 11.2×, and 285.1× less energy consumption than the dense mode in Alexnet CONV3, Alexnet FC1, and Wiki-Vote Matrix Multiplication, respectively. Through the matching type $[a - a]$, MOSDA avoids redundant data movement from the global buffer to PEs. According to the post-layout simulation, the sparse mode's energy efficiency is 1507.8 inferences/J for sparse Alexnet. The sparse mode achieves 2.4× better overall energy reduction against the dense mode, which is 624.1 inferences/J. Fig. 4.10 shows the on-chip energy breakdown

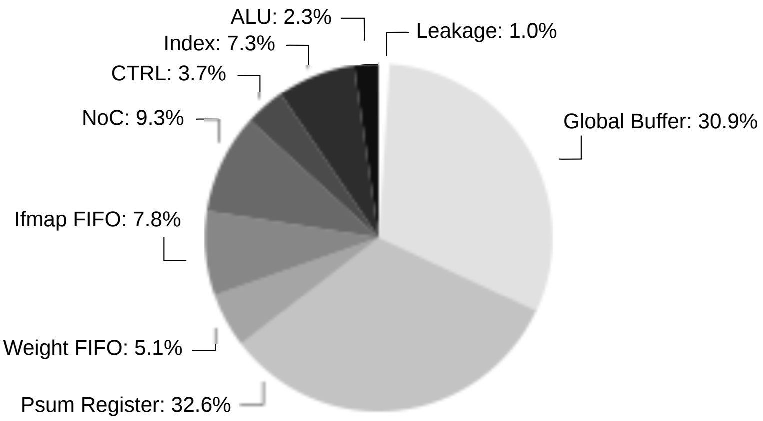


Figure 4.10: On-chip energy breakdown in sparse Alexnet.

Table 4.7: Comparison with the state-of-the-art DNN accelerators.

	EIE [23]	OuterSPACE [34]	Eyeriss v2 [27]	This Work		
Matching Type	$[s-s]$	$[a-a]$	$[s-s]$	$[a-a]$		
Technology [nm]	28	40	65	65		
Core Area	63.8mm^2	9mm^2	2695k (NAND-2)	3066k (NAND-2)		
On-chip SRAM [KB]	162	112	246	192		
Mul Num.	256	32	384	256		
Core Frequency	1200 MHz	744 MHz	200 MHz	200 MHz		
Bit Precision	4	32	8	8		
Support FC Layer	Yes	Yes	Yes	Yes		
Support CONV Layer	No	No	Yes	Yes		
Sparse Model			Alexnet	Alexnet	ResNet-50	MobileNet v2
Throughput [Inference/sec]	-	-	278.7	315.3	493.1	2764.4
Energy Efficiency [Inference/J]	-	-	664.6	1422.3	839.9	5574.6

when processing all layers in sparse Alexnet. 42.6% of the energy is consumed in the access to the psum register due to its complex read-write logic. The global buffer consumes 30.9% of the energy. Crossbars to transmit data inside the PE lead to 9.3% energy overhead. The control logic including the COO format convertor consumes 3.7% of the overall energy consumption. Leakage in MOSDA consumes 1.0% of the overall energy consumption. As a cost to ensure $[a-a]$ matching type, reuse among ifmap channels is hard to be utilized by PEs, which brings an additional number of global buffer accesses. Comparing with the architecture in Chapter 3, MOSDA requires additional on-chip memory to store the coordinate information and the crossbar overhead for the irregular data access pattern. Therefore, the dense mode of MOSDA suffers 1.4× larger energy overhead than the architecture in Chapter 3 in Alexnet.

Fig. 4.9 (a) shows MOSDA throughput enhancement from the dense mode to the sparse mode in several layers in MobileNet. The sparse mode achieves 1.2×, 1.2× and 1.5× less time consumption than the dense mode in MobileNet pointwise conv layer 2, pointwise conv layer 13, and pointwise conv layer 14, respectively. The data transmission time between the global buffer and PEs dominates the time consumption. Among all layers in dense MobileNet, data transmission between the global buffer and PEs consumes 83.0% of the on-chip time consumption. In sparse MobileNet, data transmission between the global buffer and PEs comes to be 95.5% of the on-chip time consumption. The sparse mode achieves 1.3× better overall time reduction against the dense mode when processing MobileNet.

Fig. 4.9 (b) introduces the energy efficiency of the sparse mode under various benchmarks. By considering data reuse and space reuse, energy consumption has also been reduced. The sparse mode achieves 1.2×, 1.2× and 1.4× less energy consumption than the dense mode in MobileNet pointwise conv layer 2, pointwise conv layer 13, and pointwise conv layer 14, respectively. The sparse mode's energy efficiency is 5173.9 inferences/J for sparse MobileNet, which achieves 1.2× better overall energy reduction against the dense mode as 4206.4 inferences/J when processing dense MobileNet. The dense mode of MOSDA suffers 1.1× larger energy overhead than the architecture in Chapter 3 in MobileNet.

Three state-of-the-art DNN accelerators [23, 27, 34] are compared with MOSDA in Tab. 4.7. EIE [23] and OuterSPACE [34] are specialized accelerators designed for FC layers and matrix multiplications. Eyeriss v2 [27] is an energy-efficient sparse CNN accelerator which is able to process both FC layers and CONV layers. In order to exploit more data reuse with PEs, Eyeriss v2 has the $[s - s]$ matching type in its scratchpad memories in each PE. The evaluation results on several sparse neural networks are introduced in Tab. 4.7. Based on the gate-level simulation of sparse Alexnet, the average throughput of MOSDA based $[a - a]$ dataflow when processing the sparse Alexnet is 315.3 inferences/second on average, which achieves 1.1× better improvement with 256 multipliers than the $[s - s]$ dataflow with 384 multipliers. That is because the matching type $[a - a]$ allows MOSDA to skip zero data more efficiently, thereby enhancing throughput. Similarly, although MOSDA suffers from a large accumulation overhead, it still achieves 1507.8 inferences/J, which has 2.1× better energy efficiency than the $[s - s]$ processing dataflow in sparse Alexnet. The reason is divided into two parts. First, MOSDA does not store many weights within registers in each PE, which reduces the access energy to each register. Second, $[a - a]$ processing dataflow guarantees all loaded data in registers generate meaningful results, thereby reducing the number of data movement. We further utilize sparse Resnet-50 [74] and sparse MobileNet v2 [75] with width multiplier of 1.0 and input size of 224×224 with data from ImageNet dataset to prove the effectiveness of MOSDA. According to post-layout simulations, MOSDA achieves 493.1 inferences/second and 839.9 inferences/second when processing Resnet-50 and MobileNet v2, respectively.

MOSDA achieves 2764.4 inferences/J and 5574.6 inferences/J when processing Resnet-50 and MobileNet v2, respectively.

This chapter mainly focuses on the sparsity within one frame, which is also called spatial sparsity. Besides the spatial sparsity, there is also temporal sparsity among frames [76]. If data rarely change over time, data is regarded as temporarily sparse. A large number of data movement is able to be reduced by utilizing temporal sparsity. MOSDA is also available for the temporal sparsity since the matching type over frames is also [a-a].

4.6 Summary

This chapter proposes the processing property for sparse DNN as the matching type for the PE utilization issue due to the irregular data access pattern. Focusing on three processing properties as the date reuse, the space reuse and the matching type, this chapter proposes an efficient sparse DNN hardware accelerator called MOSDA, which speeds up the processing by skipping zero data without complex control logic in the sparse DNN processing. According to a case study, MOSDA outperforms the state-of-the-art CNN accelerator with 1.1× time reduction and 2.1× energy reduction in sparse Alexnet.

Chapter 5

BNN Accelerator for Small-Scale Processing with Regular Access Pattern

5.1 Introduction

Several studies have explored DNN models that reduce the energy and time consumption according to reducing the amount of operations [2, 68]. BNN is one typical model among them [77]. In a BNN, all weights, input feature map (ifmap) pixels, and output feature map (ofmap) pixels are represented by +1 or -1 pixels. Compared to a traditional 16-bit or 32-bit data representation, BNN significantly reduces the computational effort required for prediction, thus reducing the energy consumption and time required for data movement. By transforming floating-point operations into binary calculations, many studies have designed a series of hardware accelerators for BNN inference using XNOR-popcount instead of a traditional Multiply-and-Accumulate operation [12, 78–83].

Unlike general-purpose processing, the data access pattern in BNNs is regular and can be quantitatively described by DNN properties proposed in previous chapters. Moreover, the data scale required for BNNs is usually small enough that all data can be stored on-chip to reduce the cost of data movement [78, 79]. In this chapter, we propose a hardware design with an energy-efficient memory system for BNNs.

Several studies have designed a Processing-in-Memory (PIM) based BNN accelerators [78, 82]. By tightly coupling compute units and storage units, PIM-based BNN accelerators can achieve great advantages in terms of low energy and high throughput. However, due to the tight coupling, it is difficult to handle various computational tasks efficiently with one PIM accelerator. For scalability, several works exploit dataflow processing to ensure that a BNN accelerator is capable of handling different data processing tasks [12, 79–83]. In order to maintain the versatility of the accelerators, BNN accelerators use a crossbar structure for on-chip data transmission. However, due to the energy consumption and area overhead proportional to the squared size of a crossbar structure, previous BNN accelerators are difficult to support massively parallel computation to increase the throughput as the PIM-based one. Therefore, one challenge of BNN accelerators is to design a processor that can handle multiple computational tasks without a crossbar architecture for data transmission.

Expensive data movements often dominate the energy consumption of the entire computing architecture for DNN processing [2]. Therefore, the memory system is a major component in edge BNN processors that dominate their entire energy consumption and performance. Ref. [12] shows that a memory hierarchy consumes more than 90% of the total energy consumption. Therefore, another challenge is to design the energy-efficient on-chip memory hierarchy for BNN exploiting reuse in matrix operations, since all data for BNN processing can be stored on-chip.

Based on the two challenges above, this thesis makes the following contributions as the fourth part,

1. This chapter designs a two-level on-chip memory system to reduce the energy consumption of a BNN accelerator. Unlike previous accelerators that use a single global buffer, this chapter explores the number of levels in memory hierarchy that are energy-efficient for matrix multiplication and convolution, respectively.

2. Based on the trade-off between memory capacity and energy efficiency, this chapter proposes a hybrid memory system that employs an energy-efficient low-voltage memory and a conventional 6T-SRAM into the L1 on-chip buffer and the L2 buffer, respectively. Since processing elements frequently access the energy-efficient L1 buffer, the energy consumption can be effectively reduced. Since the L2 buffer consists of 6T-SRAM with high area-density, it reduces the area requirement of on-chip memories.

3. This chapter proposes BNN dataflows for convolutions (CONVs) and fully-connected layers (FCs), respectively, to avoid crossbar structures. Without a crossbar structure, the proposed structure can significantly reduce the amount of logics required, while reducing the amount of energy required.

The rest of this chapter is organized as follows. Section 5.2 introduces BNN processing and presents a motivational example for the proposed hybrid memory structure. Section 5.3 presents a detailed architecture of the proposed hybrid memory system. Section 5.4 analyzes experimental results. Section 5.5 concludes this chapter.

5.2 Background

5.2.1 BNN Processing

Given convolution shapes in Tab. 3.1, the operation of a binary convolutional layer can be expressed by:

$$\begin{aligned}\mathbf{O}[z][u][x][y] = \mathrm{bin}_{-1,1}\Bigg(\sum_{k=1}^{C_\mathrm{i}}\sum_{i=1}^{W_\mathrm{w}}\sum_{j=1}^{H_\mathrm{w}}\Big(&\mathbf{I}[z][k][Ux+i][Uy+j]\\ &\times\mathbf{W}[u][k][i][j]\Big)+\mathbf{B}[u]\Bigg),\end{aligned} \tag{5.1}$$
$$1 \le z \le N,\ 1 \le u \le C_\mathrm{o},\ 1 \le x \le W_\mathrm{o},\ 1 \le y \le H_\mathrm{o},$$
$$H_\mathrm{o} = (H_\mathrm{i} - H_\mathrm{w} + U)/U,\ W_\mathrm{o} = (W_\mathrm{i} - W_\mathrm{w} + U)/U,$$

where weights, ifmap pixels, and ofmap pixels are represented in binary form: $\mathbf{W} \in \{-1,1\}^{C_\mathrm{i}\times C_\mathrm{o}\times H_\mathrm{w}\times W_\mathrm{w}}$, $\mathbf{I} \in \{-1,1\}^{N\times C_\mathrm{i}\times H_\mathrm{i}\times W_\mathrm{i}}$, and $\mathbf{O} \in \{-1,1\}^{N\times C_\mathrm{o}\times H_\mathrm{o}\times W_\mathrm{o}}$. **B** is a full-precision bias. The binarization ($\mathrm{bin}_{-1,1}$) is a behavior of a signum function, which converts data less than zero to -1 and data greater than or equal to zero to 1.

If we convert the expression {-1,1} to {0,1} and incorporate the addition of bias into the binarization, we can further reduce the area and energy consumption of the operation by replacing a traditional multiplication with the XNOR operation. As a result, we can simplify the operation cost by expressing a convolution in BNN as follows.

$$\begin{aligned}\mathbf{O}[z][u][x][y] = \sum_{k=1}^{C_\mathrm{i}}\sum_{i=1}^{W_\mathrm{w}}\sum_{j=1}^{H_\mathrm{w}}\Big(&\mathbf{I}[z][k][Ux+i][Uy+j]\\ &\otimes\mathbf{W}[u][k][i][j]\Big),\end{aligned} \tag{5.2}$$

where $\otimes$ is the XNOR operation.

5.2.2 Motivational Example

Comparing with SRAM-based memory, an important fact about hybrid memory systems is to exploit the non-linear relationship between memory capacity and the number of memory accesses which is discussed in Chapter 2. Assuming that only the memory capacity decreases while other parameters such as readout energy and write speed remain

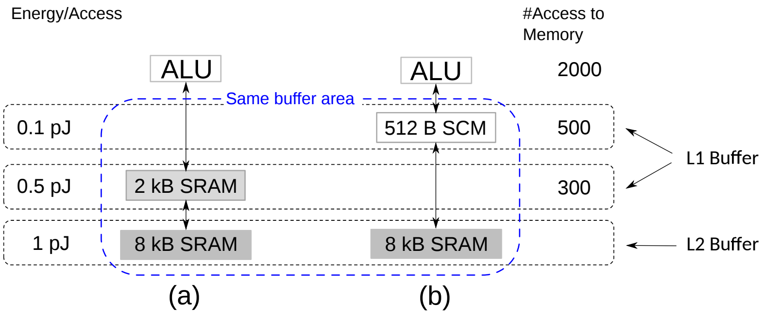


Figure 5.1: Two different memory architecture in the same on-chip area: (a) an SRAM-based memory system, (b) a hybrid memory system.

constant, the number of memory accesses usually increases slowly. Thus, replacing area-efficient SRAM with energy-efficient SCM with a fixed-area constraint can reduce the energy consumption.

For unpredictable general-purpose processing, a smaller L0 cache can be added to a hybrid memory hierarchy to reduce overall energy consumption for the proposed memory architecture in Chapter 2. In order to reduce the cost of data movement in BNN, we still use the concept of a hybrid multi-level memory hierarchy. Unlike Chapter 2, the data access pattern in BNN is fixed and explicit, and the small data scale allows us to use an on-chip memory hierarchy to store all data. In this chapter, we will design an energy-efficient memory hierarchy based on processing properties.

We design a 2-level memory architecture for BNN processing. The 2-level architecture does not only take advantage of the access efficiency of a small-capacity memory, but also to reduce the cost of data movement by using reuse in multiple steps.

Fig. 5.1 shows an example of two different memory configurations with SRAM-based memory and SCM-based memory in (a) and (b), respectively. The energy consumption per access and a corresponding number of memory accesses are written in the figure. It should be mentioned that we assume all memory systems occupy the same on-chip area.

Since an SRAM-based memory consumes more energy consumption than an SCM-based memory, we replace SRAM with SCM in the L1 buffer. When we replace SRAM with lower-area density SCM, the capacity is decreased, hence, the number of required memory accesses increases at the same time. If we can make good use of the relationship between memory capacity and the number of memory accesses, the energy overhead caused by the additional number of memory accesses can be less than the energy reduction caused by implementing the SCM memory to the L1 buffer. Here is a motivational example to explain the idea. The energy efficiency of the SCM L1 buffer is 3 × better than that of the SRAM L1 buffer with the same capacity. Due to low area density, the SCM buffer

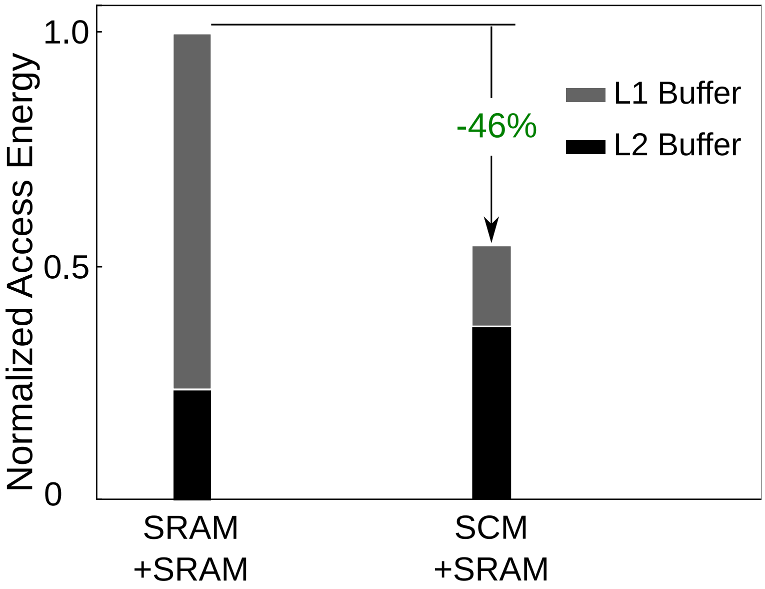


Figure 5.2: The normalized energy result of an example architecture.

next to the Arithmetic Logic Unit (ALU) suffers from 4 × less capacity than SRAM one. Therefore, the access cost to the 2 kB SRAM-L1 buffer is 5 × more energy consuming than the access cost to the 512 B SCM-L1 buffer. With the replacement of SRAM with SCM, the number of L2 buffer accesses is increased by 200 × with the regress of the L1 buffer capacity.

As a result, Fig. 5.2 introduces that the SCM L1 buffer has a larger energy reduction than the SRAM one on the left of the figure. The hybrid memory system achieves 46% less energy consumption than the SRAM-based memory system.

5.2.3 Related Work

Several accelerators are proposed to enhance BNN processing efficiency. Previous works such as Brein [78] and Ref. [82] design PIM-based accelerators. PIM-based BNN accelerators have great energy improvement and throughput enhancement. However, in the face of complex and changing processing tasks, PIM is limited by its fixed processing architecture. On the other hand, previous works propose dataflow processing architectures that exploit data reuse [12, 80, 83]. Most previous studies have used SRAM as on-chip primary memories [12]. Some studies have noted the energy consumption advantages of SCM [80, 83]. They replace all SRAMs with SCMs in anticipation of energy reduction. However, SCM-based accelerators usually result in a large area overhead due to the low area efficiency of SCM macros. Based on the previous works above, this chapter proposes a hybrid memory architecture that consists of a more energy-efficient but area-consuming

SCM as L1 buffer and an area-efficient SRAM as L2 buffer. The key focus is to utilize data reuse in two steps in order to reduce the cost of data movement.

5.3 Proposed Architecture

In this section, we first propose two dataflows for matrix multiplication and convolution respectively. The proposed dataflows guarantee a simple logic to control data movement of weights and ifmap pixels. Furthermore, this chapter proposes a BNN accelerator with a hybrid 2-level on-chip buffer system.

5.3.1 Processing Dataflows

In order to avoid a complex logic for the data transmission of ifmaps and weights, processing dataflows for matrix multiplication and convolution should follow two requirements. First, both dataflows follow the matching type $[a - a]$, which makes sure that all loaded weights multiply with all loaded ifmap pixels in ALUs. According to this, all logics for the transmission of ifmaps and weights can be shared when processing convolution tasks and matrix multiplication tasks. Second, in order to further reduce the cost of data movement in the hardware implementation for the dataflow, data reuse should also be exploited as much as possible to reduce the number of memory accesses and memory capacity.

Algorithm 5 introduces the DNN processing dataflow which satisfies both two requirements. The fifth row introduces the range of an ALU subtask inside one Processing Element (PE). The fourth and fifth rows introduce the range of the subtask among all PEs. The third, fourth, and fifth rows introduce the range of the L1 buffer subtask. The second, third, fourth, and fifth rows introduce the range of the L2 subtask. The dataflow accesses ifmap pixels concurrently on δ_2 rows as shown in the fifth row. Similarly, in order to maintain ALU utilization among PEs, the dataflow is able to process ifmap pixels from γ columns and δ_1 rows within ALUs. Parallel processing among ofmap channels is disabled for convolution in order to avoid the capacity overhead caused by partial sums. Ifmap pixels are multicasted to all weights in one channel to ensure LDR within 2D CONV among PEs. The dataflow finishes operations to increase reuse among ifmap channels within the L1 buffer as shown in the second row for both convolutions and matrix multiplications. Therefore, both operations utilize the same logics to accumulate partial sums. In the end, the rest of data reuse is utilized with the L2 buffer.

Algorithm 5 Processing dataflow for convolutions and matrix multiplications. In convolutions, β_1 and β_2 are set to 1.

```
Require: $W, I$
Ensure: $O$
 1: Initialization;
 2: for $u_1 \in [1 : C_o/\beta_1/\beta_2]$, $x_1 \in [1 : W_o/\gamma]$, $y_1 \in [1 : H_o/\delta_1/\delta_2]$ do
 3:     for $k \in [1 : C_i]$ do
 4:         for $u_2 \in [1 : \beta_1]$, $x_2 \in [1 : \gamma]$, $y_2 \in [1 : \delta_1]$ in parallel do
 5:             for $u_3 \in [1 : \beta_2]$, $y_3 \in [1 : \delta_2]$, $i \in [1 : W_w]$, $j \in [1 : H_w]$ in parallel do
 6:                 $\mathbf{O}[u_1u_2u_3][x_1x_2][y_1y_2y_3] += \mathbf{I}[k][x_1x_2+i][y_1y_2y_3+j] \times \mathbf{W}[u_1u_2u_3][k][i][j]$;
 7:             end for
 8:         end for
 9:     end for
10: end for
11: Return $O$
```

5.3.2 Hardware Architecture

The architecture overview is shown in Fig. 5.3. The L2 buffer should have enough capacity to store all the data for ifmaps, weight, and offmaps. The L1 buffers store data which can be accessed in a near future to exploit the local data reuse and the local space reuse. The L1 capacity is determined by the trade-off consideration on the amount of the reuse and the amount of L1 and L2 access energy. From our evaluation results, the capacity of the L2 SRAM buffer is chosen to be 192 kB. At the same time, 3 kB L1 buffers are designed to store data within 2D CONV. In the case of multiple PEs, SCM buffers sequentially supply each PE with required weights, ifmaps, and partial sum (psum) results. The architecture sequentially stores psum results from PEs back into an ofmap buffer after all operations in PEs have been executed. According to Algorithm 5, the number of PEs can be scaled from 1 to $\gamma\delta_1$. The architecture contains 30 PEs to enhance the throughput for BNN processing in a case study. The PE selector transmit data once γ and δ_1 are determined by target convolution shapes. Ifmap pixels and weights have a 1-bit representation, while psums have a 8-bit fixed-pointed representation. The ifmap register, the weight register, and the ofmap register are 256 bit, 256 bit, and 2048 bit, respectively.

Multiplier Configuration

In order to exploit data reuse and space reuse as much as possible, the dataflow further processes convolution ifmap channel by ifmap channel in the C_i dimension as an inner loop as shown in the third row of Algorithm 5. The convolution is processed within one 2D CONV, while the matrix multiplication is processed among multiple ofmap channels and multiple batches. Please note that the FC layer is a special case of matrix multiplications.

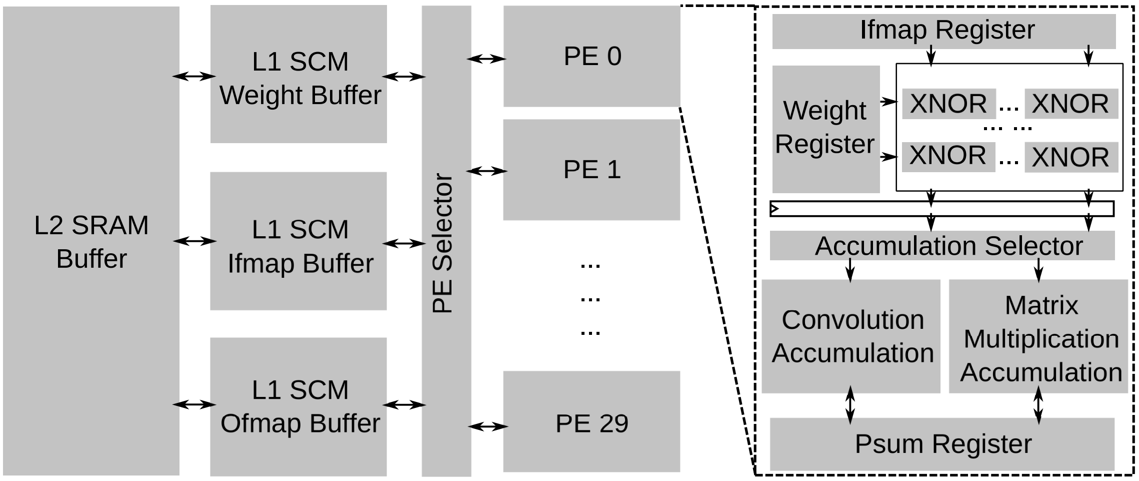


Figure 5.3: The architecture overview. The processing task within one clock cycle in one PE always belong to $[a - a]$ matching type.

63 XNOR multipliers are arranged in one PE to maintain ALU utilization. The data transmission to multipliers and the PE selector are controlled by a control logic. The control logic of multipliers is prepared for 8 different situations as matrix multiplications and convolutions with a weight size 1x1, 2x2, 3x3, 5x5, 7x7, 9x9, and 11x11. As an example, for parallel processing of CONV layers with a weight size 3x3, 63 XNOR-based multipliers are configured into 9×7. 7 multipliers in one same column share the same weight, and 9 multipliers in one same row share the same ifmap. For parallel processing of CONV layers with a weight size 5x5, 63 XNOR-based multipliers are configured into 25×2. 2 multipliers in one same column share the same weight, and 25 multipliers in one same row share the same ifmap. The power supply of remaining 13 multipliers are turned off to reduce energy consumption. For parallel processing of FC layers, 63 XNOR-based multipliers are configured into 9×7. 7 multipliers in one same column share the same ifmap pixel, and 9 multipliers in one same row share the same weight.

Accumulation Configuration

We realize the psum accumulation with an addition network. Accumulations inside one PE for convolution and matrix multiplication are realized with different addition networks. Fig. 5.3 introduces a psum accumulation logic, which is decomposed into a convolution accumulation logic and a datapath for matrix-multiplication results. An accumulation selector is utilized to choose a required accumulation logic for matrix multiplications and different weight sizes in convolutions.

Fig. 5.4 introduces the accumulation logic inside one PE for convolution with weight size as 2x2. Ifmap pixels with the coordinate of (2, 1) and (1, 1) are sent in parallel from

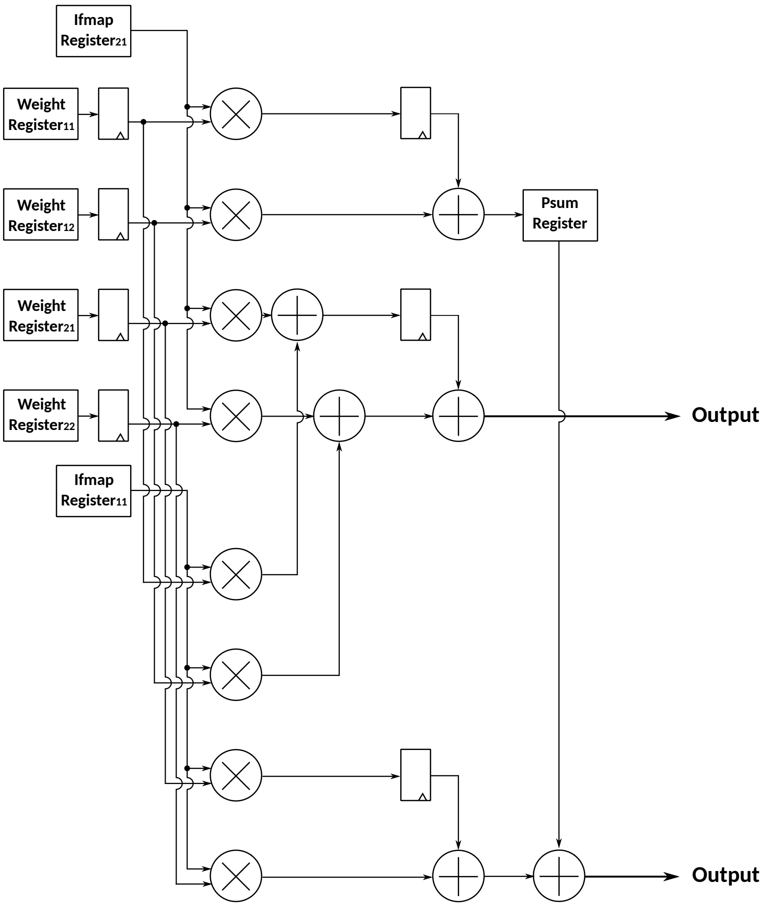


Figure 5.4: PE array architecture with fetched ifmap pixels equal to 2. The weight size is assumed to be 2 × 2.

ifmap register files 21 and 11. All weights inside 2D CONV are store in registers waiting for multiplications. We exploit local space reuse in 2D CONV within one clock cycle, add related psums and send it to psum register files. Psums which are calculated into final results are sent directly to L1 ofmap buffer. This reduces both the number of accesses and the required psum-register capacity.

5.4 Experimental Results

In this section, we verify the efficiency with 30 PEs by pre-layout cycle-accurate gate-level simulations under a 65-nm Silicon-on-Thin-Box (SOTB) process with a 1.2 V supply voltage. The bandwidth between L2 buffer and L1 buffer transmits 1890 bits once for weights, ifmap pixels, and psums. The bandwidth between L1 buffers and one PE transmits 315 bits once for weights, ifmap pixels, and psums. The specifications are summarized in Tab. 5.1. The peak throughput is defined as the total throughput of XNOR multipliers.

Table 5.1: Hardware Specifications.

Technology	65nm SOTB
Gate Count (Logic Only)	230k gates (NAND-2)
On-chip Memory	192 kB SRAM + 11 kB SCM
Clock Frequency	200 MHz
Peak Throughput	0.4 TOPS
Filter Size	1^2, 2^2, 3^2, 5^2, 7^2, 9^2, 11^2
Ifmap Size	All
Processing Type	CONV and FC (Matrix Multiplication)

Tab. 5.2 summarizes the energy of major logic components in a 1.2 V supply voltage. All energy values except for on-chip SRAM are based on average values for one operation during overall simulations. The energy consumption of the access to an L2 SRAM buffer is collected from SRAM model in CACTI Ver. 7.0 [15] under a 65-nm process with a 1.2 V supply voltage.

The PE array and SCM-based L1 buffer are evaluated from pre-layout simulations. In order to verify the validity of the energy analysis through simulations and make a fair comparison with other state-of-the-art accelerators, we did a gate-level simulation using NC-verilog to generate Value Change Dump (VCD) file. We further convert VCD to toggle information, which is dumped to Switching Activity Interchange Format (SAIF). Finally, the energy consumption is estimated from the number of toggles. The energy and area of L2 SRAM buffer part are collected from CACTI model. Based on code tracing logs, we calculate access activities of SRAM-based L2 buffer. Combining activities and physical SRAM data in Tab. 5.2, this chapter simulates the energy consumption of the overall performance.

The overall logic gate count is 230k NAND-2 gates. Fig. 5.5 introduces the area breakdown. The L2 SRAM buffer accounts for 82.1% of the total core area. The L1 SCM buffers consisting of an ifmap buffer, a weight buffer, and an ofmap buffer account for 11.3%. The control logic for data transmission of buffers consumes 1.9% of the total core area. The psum register files in 30 PEs account for 2.3%. The weight register files account for 1.3%. The ifmap register files in 30 PEs account for 1.1%.

This chapter simulates the energy consumption of the proposed architecture and an SRAM-based architecture under various benchmarks including convolution layers and FC layers from pre-layout simulations. The SRAM-based architecture has the same architecture as the proposed one while its L1 buffers are made of SRAM. According to the high area density of SRAM, the capacity of each L1 SRAM buffer is as high as 12 kB which can exploit data reuse among ifmap channels. The increase of SRAM L1 capacity

Table 5.2: Related energy consumption in convolution operations.

Component	Energy / Access [pJ]
1-bit XNOR Multiplier	0.002
8-bit Adder	0.01
1-bit Register	0.01*
3 kB SCM	0.09*
192 kB SRAM	1.27*

*: Average energy consumption of read and write.

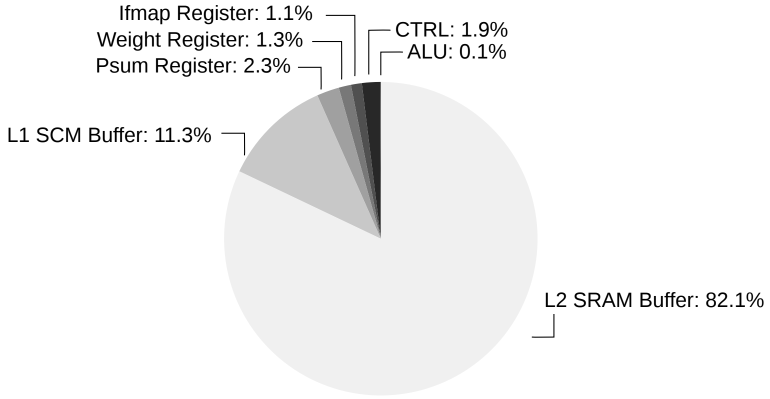


Figure 5.5: Area breakdown.

leads to the reduction in the number of L2 accesses, but also the energy overhead for the L1 access cost.

The benchmark consists of convolution layers and FC layers in Alexnet [56] and Mobilenet [53]. Fig. 5.6 introduces the energy efficiency of two architectures, which the X axis is the layer number in Alexnet. Through the matching type $[a - a]$, the proposed architecture avoids crossbar logics for data transmission of ifmap pixels and weights. The left-hand bars show the BNN accelerator with an SRAM-based design. The right-hand bars show a BNN accelerator with a proposed hybrid memory system, where L1 buffers

Table 5.3: Comparison with state-of-the-art BNN accelerators.

	XNOROBIN [12]	VLSIC2020 [82]	ChewBaccaNN [83]	This Work
Process	65nm	10nm	28nm	65nm
Maturity	Layout	Silicon	Layout	Pre-layout
Voltage	1.2V	0.37V	0.4V	1.2V
#Muls (XNOR)	1792	131072	112	1890
Kernel Support	1-7	2	1,3,5,7	1,2,3,5,7,9,11
Frequency	476 MHz	13 MHz	154 MHz	200 MHz
Memory	430 kbit SRAM	161 kB 8T Register	153 kB SCM	192 kB SRAM + 11 kB SCM
Area	368k NAND-2 (0.54 mm^2)	0.39 mm^2	0.7 mm^2	230k NAND-2
Power	32 mW	5.6 mW	0.7 mW	8 mW
Throughput	0.7 TOPS	3.4 TOPS	0.028 TOPS	0.4 TOPS
Throughput/Power	22 TOPS/W	617 TOPS/W	28 TOPS/W	38 TOPS/W

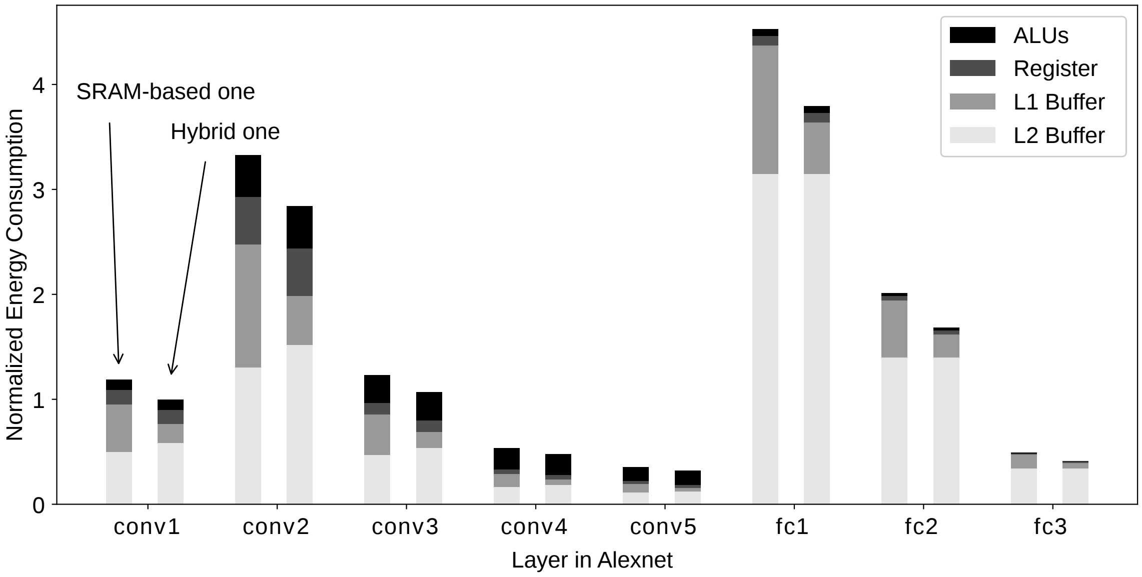


Figure 5.6: Normalized energy consumption for an SRAM-based architecture and a hybrid memory architecture when processing Alexnet.

and registers are made of SCM, which has only one-third of the effective capacity than SRAM one due to SCM low area density. The hybrid memory-based energy efficiency is 1.2 × higher than that of SRAM architecture among CONV layers in Alexnet.

The energy reduction is mainly due to an energy advantage of SCM, and the energy overhead of the increased number of memory accesses due to the capacity reduction is not significant. According to the capacity decrease of SCM L1 buffer, the hybrid memory architecture has 15.2% energy overhead of the increased number of L2 buffer accesses for the whole accelerator. Results for all layers when processing Alexnet show that the overall energy efficiency of the accelerator with the hybrid memory system is 1.3 × more efficient than the SRAM-based architecture.

Fig. 5.8 shows the on-chip energy breakdown of the hybrid-memory architecture when processing CONV layers in Alexnet. 40.8% of the energy is consumed in the access to registers inside PEs. The L1 SCM buffers consume 16.3% of the energy. The L2 SRAM buffers consume 30.2% of the energy. Fig. 5.7 introduces the energy efficiency in Mobilenet. By considering data reuse and space reuse, energy consumption has also been reduced. The hybrid architecture achieves 1.2× less energy consumption than the SRAM-based architecture in MobileNet.

Three state-of-the-art DNN accelerators [23, 27, 34] are further compared with the proposed architecture in Tab. 5.3. XNOROBIN [12] and ChewBaccaNN [83] are specialized accelerators designed for BNN processing. Ref. [82] is an energy-efficient Processing-in-

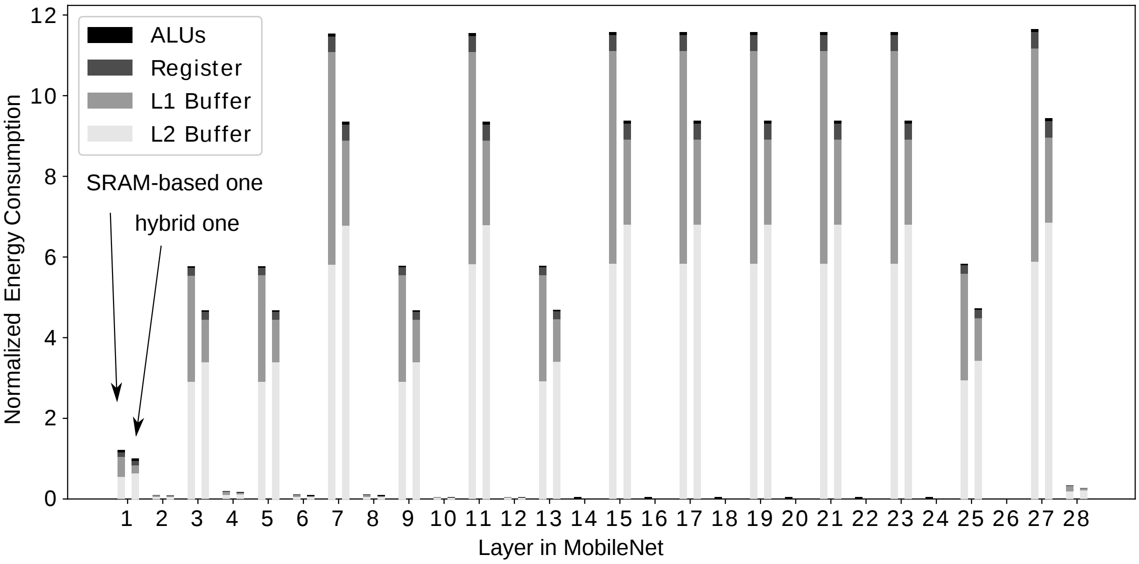


Figure 5.7: Normalized energy consumption for an SRAM-based architecture and a hybrid memory architecture when processing Mobilenet.

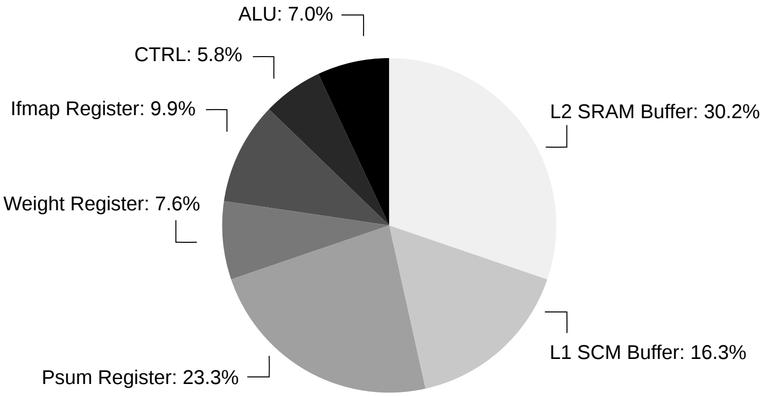


Figure 5.8: On-chip energy breakdown for all layers in Alexnet.

Memory BNN accelerator which is able to process convolution with kernel size equal to 2. The evaluation results on Alexnets are introduced in Tab. 5.3. The proposed architecture has a large throughput disadvantage compared to a PIM accelerator in Ref. [82]. The performance advantage of the PIM accelerator comes at the cost of their inability to handle multiple tasks. The proposed architecture achieves energy efficiency as 38 TOPS/W, which has 1.7× and 1.4× better energy efficiency than XNOROBIN and ChewBaccaNN, respectively. The reason is divided into two parts. First, the proposed architecture avoids crossbar architecture which guarantees the scalability and the efficiency of the acceleration architecture. Second, the hybrid memory system balance the area overhead of storing all data on-chip and the energy overhead to SRAM, thereby reducing the cost of data movement.

5.5 Summary

Focusing on the relationship between the memory capacity and the number of required memory accesses, this chapter proposes a hybrid memory system to reduce on-chip data movement for BNN processing. According to the specialized dataflows for matrix multiplication and convolution, the proposed hardware achieves energy-efficient processing without a crossbar logic in DNN processing. According to a case study, the proposed architecture outperforms the state-of-the-art CNN accelerator with 1.4× energy efficiency in binary Alexnet.

Chapter 6

Conclusion

6.1 Summary of This Thesis

In data processing, the cost of data movement often dominates the hardware performance. The cost of data movement in memories is determined by the memory capacity and the number of memory accesses in the memory hierarchy. This thesis chooses representative types of processing tasks as general-purpose processing and DNN processing. This thesis discusses data properties that determine the data movement in each processing task. This thesis designs four hardware designs to reduce the cost of data movement in different data processing tasks.

In general-purpose processing, based on the non-linear relation between cache miss rate and capacity, a hybrid cache design is discussed to show SCM-based design has the energy advantage over SRAM-based design. SCM has a low processing speed, a low area density but inherent energy efficiency. The key idea is to utilize SCM rather than 6T-SRAM as the L0 cache exploiting the non-linear relationship. The 2-level hybrid cache architecture exhibits 68% smaller energy consumption than the conventional 6T-based cache architecture in the simulation. When an ultra-low-power oriented application is targeted, the CPU requires near-threshold operation where 6T-SRAM cannot operate correctly due to its vulnerability against noise and process variability. If we utilize 10T SRAM for near-threshold operation, due to the SRAM read operation by a strong sense amplifier, it is still not energy-efficient enough. In contrast, if the CPU is under the near-threshold region, SCM operates well with the same supply voltage as the CPU logic.

In DNN processing, the data movement among memories and processing elements consumes most of time, energy, and area. At the same time, the number of operations in DNN processing is usually much larger than the number of hardware resources. The whole processing task should be divided into subtasks for fine-grained quantitative evaluation. By defining reuse in each subtask, it becomes possible to calculate the cost of data movement. This thesis proposes two hardware properties, local data reuse, and local space reuse to describe the cost of data movement in the memory hierarchy. Based on properties, this thesis proposes a CNN processing dataflow and a new set of hardware metrics to analytically estimate energy, throughput, and area from the processing dataflow. According to the analytical evaluation of various processing dataflows and various processing environments, the proposed evaluation metrics effectively evaluate the processing dataflow without hardware implementation. Designers are able to use the evaluation metrics to greatly reduce the workload of hardware designs.

This thesis proposes a processing property as the matching type for the PE utilization issue in a sparse DNN processing. The key idea is to keep loaded processing tasks within ALUs as [a-a] matching type in order to directly infer the number of operations from the number of loaded non-zero ifmap pixels and the number of loaded non-zero weights. As a result, this thesis can easily arrange weights and ifmap pixels required for massively parallel processing. Focusing on processing properties, this thesis proposes an efficient sparse-DNN hardware accelerator called MOSDA, which speeds up processing by skipping zero data without a complex control logic in a sparse DNN processing. According to a case study, MOSDA outperforms state-of-the-art CNN accelerators with 1.1× time reduction and 2.1× energy reduction in sparse Alexnet.

This thesis further proposes a hybrid memory system for BNN processing. According to specialized dataflows for matrix multiplication and convolution, the proposed hardware achieves energy-efficient memory architecture for BNN processing. According to a case study, the proposed architecture outperforms a state-of-the-art binary CNN accelerator with 1.4× energy efficiency in binary Alexnet.

6.2 Future Work

In large-scale parallel processing, the energy for data transmission can be even more significant. As an example, the logic for on-chip data transmission consumes 62% of the total time consumption, 41% of the total energy consumption, and 29% of the total area for sparse Alexnet in MOSDA. If the parallel scale of a hardware is larger, the cost of data transmission will account for more. Therefore, finding data properties that reasonably describe the consumption of data transmission will help us a lot when designing accelerators.

The discussion in this thesis works well towards matrix operations between weights and ifmaps. However, emerging DNN processing tasks bring more challenges to parallel processing architectures. In recent years, DNN processing has seen the emergence of new operations such as a transformer [84] which contains four different sets of input data. In the encoder of the transformer, the data is first passed through a module called "self-attention" to obtain a feature map. In the self-attention module, each word has three different weights: Query Weight, Key Weight, and Value Weight. The transformer multiplies ifmaps with three different weights into different intermediate matrices and concatenates result matrices. This thesis proposes three processing properties which are local data reuse, local space reuse, and matching type. Those properties target processing tasks with one main operation which is either matrix multiplication or convolution. When faced with a processing task where there are multiple primary operations, the parallel data movement may become more complex. Existing processing properties should be extended to support new DNN operations such as the transformer in the future work.

Bibliography

[1] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery, "Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications," *ACM Computing Surveys*, vol. 39, no. 9, pp. 33–43, 2007.

[2] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[3] S. Mittal, "A Survey of Architectural Techniques for Improving Cache Power Efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33–43, Nov 2013.

[4] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[5] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[6] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 369383. [Online]. Available: https://doi.org/10.1145/3373376.3378514

[7] A. Parashar, P. Raina, Y. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *2019 IEEE International Symposium on*

Performance Analysis of Systems and Software (ISPASS). Los Alamitos, CA, USA: IEEE Computer Society, mar 2019, pp. 304–315. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISPASS.2019.00042

[8] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 754768. [Online]. Available: https://doi.org/10.1145/3352460.3358252

[9] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-Chip Predictor: An Analytical Performance Predictor for DNN Accelerators with Various Dataflows and Hardware Architectures," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 1593–1597.

[10] T. Wing and Z. Benjamin, *Computer Hardware/Software Architecture.* Prentice Hall, 1986.

[11] S. William., *Computer Organization and Architecture : Designing for Performance (8th Ed.).* Upper Saddle River, NJ: Prentice Hall, 2010.

[12] A. Al Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, "XNORBIN: A 95 TOp/s/W Hardware Accelerator for Ninary Convolutional Neural Networks," in *2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2018, pp. 1–3.

[13] J. L. Hennessy, D. A. Patterson, K. Asanovi, J. D. Bakos, R. P. Colwell, A. Bhattacharjee, T. M. Conte, J. Duato, D. Franklin, D. Goldberg, N. P. Jouppi, S. Li, N. Muralimanohar, G. D. Peterson, T. M. Pinkston, P. Ranganathan, D. A. Wood, C. Young, and A. Zaky, *Computer Architecture: a Quantitative Approach.* Cambridge, MA : Morgan Kaufmann Publishers, 2019.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24–36.

[15] S. J. E. Wilton and N. Jouppi, "CACTI: an Enhanced Cache Access and Cycle Time Model," *Journal of Solid State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[16] D. Nagle, R. Uhlig, T. M. Mudge, and S. Sechrest, “Optimal Allocation of On-Chip Memory for Multiple-API Operating Systems,” in *ISCA ’94 Proceedings of the 21st Annual International Symposium on Computer Architecture*. IEEE, 1994, pp. 358–369.

[17] J. M. Mulder, N. T. Quach, and M. J. Flynn, “An Area Model for On-Chip Memories and its Application,” *Journal of Solid State Circuits*, vol. 26, no. 2, pp. 98–106, Feb 1991.

[18] R. G. Dreslinski, G. K. Chen, T. Mudge, D. Blaauw, D. Sylvester, and K. Flautner, “Reconfigurable Energy Efficient Near-Threshold Cache Architectures.” in *International Symposium on Microarchitecture*. IEEE, Nov 2008, pp. 459–470.

[19] J. Lee and S. Kim, “Filter Data Cache: An Energy-Efficient Small L0 Data Cache Architecture Driven by Miss Cost Reduction,” *IEEE Transactions on Computers*, vol. 64, pp. 1–1, 01 2014.

[20] J. Kin, M. Gupta, and W. H. Mangione-Smith, “The Filter Cache: An Energy Efficient Memory Structure,” in *MICRO 30 Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*. IEEE, 1997, pp. 184–193.

[21] N. P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of A Small Fully-Associative Cache and Prefetch Buffers,” in *ISCA ’90 Proceedings of the 17th annual international symposium on Computer Architecture*. IEEE, 1990, pp. 364–373.

[22] J. L. *et al*, “UNPU: A 50.6TOPS/W Unified Deep Neural Network Accelerator with 1b-to-16b Fully-Variable Weight Bit-Precision,” *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 218–220, 2018.

[23] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “EIE: Efficient Inference Engine on Compressed Deep Neural Network,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 243–254.

[24] N. P. J. *et al*, “In-datacenter Performance Analysis of A Tensor Processing Unit,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 1–12.

[25] J. Jo, S. Kim, and I. Park, “Energy-Efficient Convolution Architecture Based on Rescheduled Dataflow,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4196–4207, Dec 2018.

[26] B. Moons and M. Verhelst, “An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, April 2017.

[27] Y. Chen, T. Yang, J. Emer, and V. Sze, “Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[28] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, “Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 1–13.

[29] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “ShiDianNao: Shifting Vision Processing Closer to the Sensor,” in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 92–104.

[30] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, “An Architecture to Accelerate Convolution in Deep Neural Networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1349–1362, April 2018.

[31] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H. Yoo, “1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications,” in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015, pp. 1–3.

[32] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, “A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications,” *2017 Symposium on VLSI Circuits*, pp. C26–C27, 2017.

[33] M. Cho and D. Brand, “MEC: Memory-Efficient Convolution for Deep Neural Network,” *CoRR*, vol. abs/1706.06873, 2017.

[34] S. P. *et al*, “A 7.3 M Output Non-Zeros/J Sparse Matrix-Matrix Multiplication Accelerator using Memory Reconfiguration in 40 nm,” in *2019 Symposium on VLSI Technology*, June 2019, pp. C150–C151.

[35] Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, and H. Yang, "STICKER: An Energy-Efficient Multi-Sparsity Compatible Accelerator for Convolutional Neural Networks in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, 2020.

[36] H. Mizuno and K. Ishibashi, "A Separated Bit-Line Unified Cache: Conciliating Small On-Chip Cache Die-Area and Low Miss Ratio." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, pp. 139 – 144, Mar 1999.

[37] H. Xu, J. Shiomi, T. Ishihara, and H. Onodera, "Maximizing Energy Efficiency of on-Chip Caches Exploiting Hybrid Memory Structure," in *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS).* IEEE, July 2018, pp. 237–242.

[38] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson, "A Sub-200mV 6T SRAM in 0.13 μm CMOS," in *2006 IEEE International Solid State Circuits Conference (ISSCC)*, 03 2007, pp. 332 – 606.

[39] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-driven Near-threshold SRAM Design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, pp. 1590 – 1598, 12 2010.

[40] Z. Sun, X. Bi, H. Li, W.-F. Wong, and X. Zhu, "STT-RAM Cache Hierarchy With Multiretention MTJ Designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 1281–1293, 2014.

[41] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Through Controlled Placement," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, pp. 1–25, 05 2016.

[42] J. Shiomi, T. Ishihara, and H. Onodera, "Area-Efficient Fully Digital Memory Using Minimum Height Standard Cells for Near-Threshold Voltage Computing," *Integration, the VLSI Journal*, 07 2017.

[43] B. Calhoun and A. Chandrakasan, "A 256kb Sub-threshold SRAM in 65nm CMOS." in *2006 IEEE International Solid State Circuits Conference (ISSCC).* IEEE, Feb 2006, pp. 2592–2601.

[44] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamanda, F. K. Grkaynak, and L. Benini, “Near-Threshold RISC-V Core with DSP Extensions for Scalable IoT Endpoint Devices,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, Oct 2017.

[45] K. Shin, W. Choi, and J. Park, “Half-Select Free and Bit-Line Sharing 9T SRAM for Reliable Supply Voltage Scaling,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, pp. 1–13, 04 2017.

[46] S. Nakamura and K. Usami, “Level Converter Design for Ultra-low Voltage Operation in FDSOI Devices,” in *The 29th International Technical Conference on Circuits/Systems, Computers and Communications*, 2014, pp. 93–96.

[47] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Chandrakasan, and V. De, “Adaptive Body Bias for Reducing Impacts of Die-to-Die and Within-Die Parameter Variations on Microprocessor Frequency and Leakage,” *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1396–1402, 2002.

[48] L. Yann, B. Yoshua, and H. Geoffrey, “Deep Learning,” *Nature*, vol. 521, no. 1, pp. 436–444, May 2015.

[49] K. Vissers, “Versal: The Xilinx Adaptive Compute Acceleration Platform (ACAP),” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’19. New York, NY, USA: ACM, 2019, pp. 83–83.

[50] Intel. (2019) Neural Compute Stick 2. [Online]. Available: https://software.intel.com/en-us/articles/OpenVINO-RelNotes

[51] Google. (2019) Edge TPU. [Online]. Available: https://cloud.google.com/edge-tpu/

[52] D. Silver, A. Huang, C. Maddison, and *et al*, “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, no. 484, p. 484489, Jan 2016.

[53] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *CoRR*, vol. abs/1704.04861, 2017.

[54] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” in *2015 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 1–9.

[55] J. Jo, S. Cha, D. Rho, and I. Park, "DSIP: A Scalable Inference Accelerator for Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 605–618, Feb 2018.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[57] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv 1409.1556*, 09 2014.

[58] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[59] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[61] S. Han, J. Pool, J. Tran, and W. Dally, "Learning Both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 1135–1143.

[62] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5MB Model Size," 2016.

[63] S. Anwar, K. Hwang, and W. Sung, "Structured Pruning of Deep Convolutional Neural Networks," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, Feb. 2017.

[64] T. Yang, Y. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6071–6079.

[65] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A Systematic DNN Weight Pruning Framework Using Alternating Direction Method of Multipliers," in *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018, pp. 191–207.

[66] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating Deep Convolutional Networks using low-precision and sparsity," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2861–2865.

[67] O. Moreira, A. Yousefzadeh, F. Chersi, A. Kapoor, R. . Zwartenkot, P. Qiao, G. Cinserin, M. A. Khoei, M. Lindwer, and J. Tapson, "NeuronFlow: A Hybrid Neuromorphic Dataflow Processor Architecture for AI Workloads," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 01–05.

[68] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[69] Y. Saad, "SPARSKIT: A Basic Tool Kit for Sparse Natrix Computations - Version 2," 1994.

[70] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 2740, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080254

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[72] J. Leskovec and R. Sosič, "SNAP: A General-Purpose Network Analysis and Graph-Mining Library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[73] M. Zhu and S. Gupta, “To Prune, or Not to Prune: Exploring the Efficiency of Pruning for Model Compression,” *ArXiv*, vol. abs/1710.01878, 2018.

[74] T. Gale, E. Elsen, and S. Hooker, “The State of Sparsity in Deep Neural Networks,” *CoRR*, vol. abs/1902.09574, 2019. [Online]. Available: http://arxiv.org/abs/1902.09574

[75] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan, “Fast Sparse ConvNets,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2020, pp. 14 617–14 626. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.01464

[76] A. Yousefzadeh, M. A. Khoei, S. Hosseini, P. Holanda, S. Leroux, O. Moreira, J. Tapson, B. Dhoedt, P. Simoens, T. Serrano-Gotarredona, and B. Linares-Barranco, “Asynchronous Spiking Neurons, the Natural Key to Exploit Temporal Sparsity,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 668–678, 2019.

[77] M. Courbariaux and Y. Bengio, “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[78] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, “BRein Memory: A Single-Chip Binary/Ternary Reconfigurable in-Memory Deep Neural Network Accelerator Achieving 1.4 TOPS at 0.6 W,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, 2018.

[79] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, “BinarEye: An Always-On Energy-Accuracy-Scalable Binary CNN Processor With All Memory On Chip in 28nm CMOS,” 2018.

[80] F. Conti, P. D. Schiavone, and L. Benini, “XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[81] J. Yue, R. Liu, W. Sun, Z. Yuan, Z. Wang, Y. Tu, Y. Chen, A. Ren, Y. Wang, M. Chang, X. Li, H. Yang, and Y. Liu, “A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration

with 8.1 Œ Higher TOPS/mm2and 6T HBST-TRAM-Based 2D Data-Reuse Architecture," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 138–140.

[82] P. C. Knag, G. K. Chen, H. E. Sumbul, R. Kumar, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, M. Kar, S. Kim, and R. K. Krishnamurthy, "A 617 TOPS/W All Digital Binary Neural Network Accelerator in 10nm FinFET CMOS," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[83] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "ChewBaccaNN: A Flexible 223 TOPS/W BNN Accelerator," 2020.

[84] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

Publication List

Journal

1. Hongjie Xu, Jun Shiomi, and Hidetoshi Onodera, “Evaluation Metrics for the Cost of Data Movement in Deep Neural Network Acceleration," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, (under review).

2. Hongjie Xu, Jun Shiomi, and Hidetoshi Onodera, “MOSDA: On-chip Memory Optimized Sparse Deep Neural Network Accelerator with Efficient Index Matching," *IEEE Open Journal of Circuits and Systems*, 2020, in press `http://dx.doi.org/10.1109/OJCAS.2020.3035402`

3. Hongjie Xu, Jun Shiomi, Tohru Ishihara, and Hidetoshi Onodera, “On-Chip Cache Architecture Exploiting Hybrid Memory Structures for Near-Threshold Computing," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E102-A, no.12, pp. 1741–1750, Dec. 2019.

International Conference

1. Hongjie Xu, Jun Shiomi, and Hidetoshi Onodera, “On-chip Memory Optimized CNN Accelerator with Efficient Partial-sum Accumulation," *Proceedings of the 30th edition of the ACM Great Lakes Symposium on VLSI*, Sep. 2020, pp. 21–26.

2. Hongjie Xu, Jun Shiomi, Tohru Ishihara, and Hidetoshi Onodera, “Maximizing Energy Efficiency of On-Chip Caches Exploiting Hybrid Memory Structure," *Proceedings of the International Workshop on Power And Timing Modeling, Optimization and Simulation*, Jul. 2018, pp. 237–242.

3. Hongjie Xu, Jun Shiomi, Tohru Ishihara, and Hidetoshi Onodera, “A Hybrid Caching System Using SRAM and Standard-Cell Memory for Energy-Efficient

Near-Threshold Circuits," *in Workshop on Synthesis And System Integration of Mixed Information technologies*, Mar. 2018, pp. 56–61.